TASK: PV03
CDRL: A023
17 March 1995

# Organization Domain Modeling (ODM) Guidebook, Version 1.0

STARS-VC-A023/011/00

**TASK: PV03**
**CDRL: A023**
**17 March 1995**

# INFORMAL TECHNICAL REPORT
For
SOFTWARE TECHNOLOGY FOR ADAPTABLE, RELIABLE SYSTEMS
(STARS)

*Organization Domain Modeling (ODM) Guidebook*
*Version 1.0*

STARS-VC-A023/011/00
17 March 1995

Data Type: Informal Technical Data

CONTRACT NO. F19628-93-C-0130

Prepared for:
Electronic Systems Center
Air Force Systems Command, USAF
Hanscom, AFB, MA 01731-2816

Prepared by:
Unisys Corporation
12010 Sunrise Valley Drive
Reston, VA 22091

**TASK: PV03**
**CDRL: A023**
**17 March 1995**

Data Reference: STARS-VC-A023/011/00
INFORMAL TECHNICAL REPORT
*Organization Domain Modeling (ODM) Guidebook*
*Version 1.0*

**TASK: PV03**
**CDRL: A023**
**17 March 1995**

Data Reference: STARS-VC-A023/011/00
INFORMAL TECHNICAL REPORT
*Organization Domain Modeling (ODM) Guidebook*
*Version 1.0*

## Abstract

This guidebook describes the Organization Domain Modeling (ODM) domain engineering method. ODM has been developed by Mark Simos of Organon Motives to systematize key aspects of the domain modeling process and provide an overall framework for a domain engineering life cycle. ODM is intended as a highly tailorable and configurable domain engineering process model, useful for diverse organizations and domains, and amenable to integration with a variety of software engineering processes, methods, and implementation technologies. In addition to focusing on the strictly technical aspects of domain engineering, the method emphasizes the importance of clearly defining the organization context within which each domain engineering effort is conducted.

The primary purposes of the guidebook are to:

- Provide a definitive ODM reference document which promotes public understanding of the method and its applicability through in-depth descriptions of ODM concepts, processes, and workproducts.
- Provide substantial practical guidance for using the method by describing ODM activities in detail and offering workproduct templates and examples to get practitioners started.
- Provide general guidance in tailoring the method for application within a particular project or organization.

**TASK: PV03**
**CDRL: A023**
**17 March 1995**

Data Reference: STARS-VC-A023/011/00
INFORMAL TECHNICAL REPORT
*Organization Domain Modeling (ODM) Guidebook*
*Version 1.0*

**Principal Author(s):**

*Mark Simos, Organon Motives* Date

*Dick Creps, Unisys* Date

*Carol Klingler, Unisys* Date

*Larry Levine, Organon Motives* Date

**Approvals:**

Teri F. Payton 3/20/95

Program Manager *Teri F. Payton* Date

(Signatures on File)

| REPORT DOCUMENTION PAGE | *Form Approved* *OMB No. 0704-0188* |
|---|---|

| | | |
|---|---|---|
| 1. AGENCY USE ONLY *(Leave Blank)* | 2. REPORT DATE<br>17 March 1995 | 3. REPORT TYPE AND DATES COVERED<br>Informal Technical |

| | |
|---|---|
| 4. TITLE AND SUBTITLE<br>Organization Domain Modeling (ODM) Guidebook, Version 1.0 | 5. FUNDING NUMBERS<br>F19628-93-C-0130 |
| 6. AUTHOR(S)<br>M. Simos (Organon Motives), R. Creps (Unisys Corp), C. Klingler (Unisys), L. Lavine (Organon Motives) | |
| 7. PERFORMING ORGANIZATION NAME(S) AND ADDRESS(ES)<br>Unisys Corporation<br>12010 Sunrise Valley Drive<br>Reston, VA 22091-3499 | 8. PERFORMING ORGANIZATION REPORT NUMBER<br>CDRL NBR<br>STARS-VC-A023/011/00 |
| 9. SPONSORING/MONITORING AGENCY NAME(S) AND ADDRESS(ES)<br>Department of the Air Force<br>ESC/ENS<br>Hanscom AFB, MA 01731-2816 | 10. SPONSORING/MONITORING AGENCY REPORT NUMBER<br>A023 |

11. SUPPLEMENTARY NOTES

| | |
|---|---|
| 12a. DISTRIBUTION/AVAILABILITY STATEMENT<br>Distribution Statement “A” | 12b. DISTRIBUTION CODE |

13. ABSTRACT *(Maximum 200 words)*

This guidebook describes the Organization Domain Modeling (ODM) domain engineering method. ODM has been developed by Mark Simos of Organon Motives to systematize key aspects of the domain modeling process and provide an overall framework for a domain engineering life cycle. ODM is intended as a highly tailorable and configurable domain engineering process model, useful for diverse organizations and domains, and amenable to integration with a variety of software engineering processes, methods, and implementation technologies. In addition to focusing on the strictly technical aspects of domain engineering, the method emphasizes the importance of clearly defining the organization context within which each domain engineering effort is conducted.

The primary purposes of the guidebook are to:

- Provide a definitive ODM reference document which promotes public understanding of the method and its applicability through in-depth descriptions of ODM concepts, processes, and workproducts.
- Provide substantial practical guidance for using the method by describing ODM activities in detail and offering workproduct templates and examples to get practitioners started.
- Provide general guidance in tailoring the method for application within a particular project or organization.

| | |
|---|---|
| 14. SUBJECT TERMS | 15. NUMBER OF PAGES<br>338 |
| | 16. PRICE CODE |

| | | | |
|---|---|---|---|
| 17. SECURITY CLASSIFICATION OF REPORT<br>Unclassified | 18. SECURITY CLASSIFICATION OF THIS PAGE<br>Unclassified | 19. SECURITY CLASSIFICATION OF ABSTRACT<br>Unclassified | 20. LIMITATION OF ABSTRACT<br>SAR |

TASK: PV03
CDRL: A023
17 March 1995

Data Reference: STARS-VC-A023/011/00
**INFORMAL TECHNICAL REPORT**
*Organization Domain Modeling (ODM) Guidebook*
*Version 1.0*

## Table of Contents

## Table of Contents

## Table of Contents

## Table of Contents

## List of Exhibits

## List of Exhibits

## List of Exhibits

# Prologue

This document is version 1.0 of the *Organization Domain Modeling (ODM) Guidebook.* The guidebook describes the ODM domain engineering method developed principally by Mark Simos of Organon Motives, with sponsorship from the Software Technology for Adaptable, Reliable Systems (STARS) program (on behalf of the U.S. Department of Defense (DoD) Advanced Research Projects Agency (ARPA)) and the Hewlett-Packard Company.

The principal author of the guidebook was Mark Simos, with assistance from Dick Creps and Carol Klingler of Unisys and Larry Levine of Organon Motives. The authors also wish to acknowledge Patricia Collins of Hewlett-Packard for her substantial contributions to the method. In addition, James Solderitsch of Unisys and Grant Wickman of the Australian Army have made substantial contributions to the method while applying ODM on the Army STARS Demonstration Project for the US Army Communications and Electronics Command (CECOM).

The guidebook supercedes an earlier STARS draft document entitled *Organization Domain Modeling (ODM), Volume I -- Conceptual Foundations, Process and Workproduct Descriptions, Version 0.5*. The earlier report served as an initial version of an ODM reference document which addressed a wide variety of ODM topics and issues, but did not focus specifically on providing practical guidance in the use of ODM. In contrast, one of this guidebook's key objectives is to convey such knowledge and guidance. In addition, the guidebook significantly expands the scope of the earlier document to encompass the entire domain engineering process.

However, this version of the guidebook is still less complete and consistent than the authors would like. While useful in its current state, it should probably still be considered an "advanced draft." It is an interim step towards the final STARS ODM Guidebook (version 2.0), planned for release by the end of 1995. Version 2.0 will further clarify and refine the ODM process descriptions, will include a more thorough set of work product templates and worked examples, and will be re-designed to further ease comprehension of the material. Version 2.0 will also reflect ongoing lessons learned in applying the method and will address comments received from users and reviewers of version 1.0.

We thus strongly encourage trial use of ODM and solicit reader review and comments as input to the planned guidebook revision. To learn more about ODM, discuss how to obtain support in applying it, or submit comments on the guidebook, please contact either:

Mark Simos
Organon Motives
36 Warwick Road
Watertown, MA 02172
Phone: (617) 625-2170
Fax: (617) 628-6010
E-mail: simos@media.mit.edu

or

Dick Creps
Unisys
12010 Sunrise Valley Drive
Reston, VA 22091
Phone: (703) 620-7100
Fax: (703) 620-7916
E-mail: creps@stars.reston.unisysgsg.com

# Part I: Introduction

# 1.0 Guidebook Overview

This guidebook describes the Organization Domain Modeling (ODM) domain engineering method. ODM has been developed to systematize key aspects of the domain modeling process and provide an overall framework for a domain engineering life cycle. In addition to focusing on the strictly technical aspects of domain engineering, the method emphasizes the importance of clearly defining the organization context within which each domain engineering effort is conducted.

## 1.1 Purpose and Scope

The primary purposes of this guidebook are to:

- Provide a definitive ODM reference document which promotes public understanding of the method and its applicability through in-depth descriptions of ODM concepts, processes, and workproducts.
- Provide substantial practical guidance for using the method by describing ODM activities in detail and offering workproduct templates and examples to get practitioners started.
- Provide general guidance in tailoring the method for application within a particular project or organization.

The guidebook is **not** directly intended to do the following:

- Provide extensive justification and rationale for reuse and domain engineering in general or for the ODM domain engineering approach in particular.
- Define the specific role of ODM within an overall software engineering process or life cycle.
- Prescribe a full range of specific methods that can be used to support domain engineering, beyond those that are considered to be inherent or unique to the "core" ODM method, as described herein. (However, the guidebook does identify several general categories of supporting methods and describes where they are applicable within ODM.)
- Prescribe specific tools that can be used to support ODM. (However, in conjunction with the supporting methods, several categories of supporting tools are identified.)
- Compare ODM with other domain analysis, domain engineering, or reuse methods or processes (although it may be useful in helping the reader to draw such comparisons).

All of these are potential topics for future papers and reports that will complement the guidebook. In particular, ODM's relationship to the overall software life cycle and to specific techniques and tools will vary from one organization to another. These issues thus contribute to the general problem of tailoring ODM to specific contexts. Although this guidebook provides general tailoring guidance, there is a need for a companion document that provides more extensive guidance in tailoring ODM.

## 1.2 Audience

This guidebook is targeted to readers having one or more of the following roles in their organizations:

- *Program/Project Planner* — Responsible for planning the objectives, strategy, processes, infrastructure, and resources for software engineering programs or projects. Interested in incorporating systematic, domain-specific reuse into those programs/projects.
- *Reuse Advocate* — Responsible for keeping abreast of reuse concepts, technology, and trends and promoting the establishment/improvement of reuse capabilities and practices within an organization. Interested in understanding how new concepts, processes, methods, and tools can be applied to accelerate reuse adoption.
- *Process Engineer* — Responsible for defining, instantiating, tailoring, installing, automating, monitoring, administering, and evolving software engineering process models. Interested in defining reuse processes or integrating them with overall life cycle process models.
- *Domain Engineer* — Responsible for scoping and modeling domains of interest to an organization and designing and implementing collections of assets ("asset bases") that can be reused in multiple application systems. Interested in learning to apply domain engineering concepts, processes, methods, and tools.

## 1.3 Benefits

By reading this guidebook, various segments of the audience should be better able to do some or all of the following:

- Understand key ODM concepts, including:
  - The ODM process flow from Plan --> Model Domain --> Engineer Asset Base, the reasoning for this flow, and the key process steps involved
  - The need to model organization context as an integral part of domain engineering
  - The difference between system and domain modeling techniques and the distinct roles these techniques play in domain engineering
- Assess and communicate the benefits and implications of applying ODM concepts, e.g.:
  - Business problems and opportunities where domain modeling may be useful
  - Value of ODM methods and processes in relation to other software engineering methods, processes, and tools
  - Ways ODM can be tailored and integrated to meet an organization's needs
- Decide whether or not to explore ODM further

In conjunction with further reading, training, and consultation, the guidebook should enable readers to:

- Assess their organization's receptivity to applying ODM and related methods
- Initiate, plan, manage, and perform a domain engineering project using ODM

- Make technology choices for the various supporting methods required to perform a complete ODM domain engineering project

## 1.4 Guidebook Organization

The body of the guidebook is organized into three major parts:

- **Part I: Introduction**
  - *Section 1: Document Overview (this section)* — Defines the purpose, scope, and audience of the document and establishes its relationship to other reuse-related work.
  - *Section 2: ODM Background* — Describes the factors that motivated the development of ODM, the context in which it was developed, and how it has been applied.
  - *Section 3: ODM Core Concepts* — Articulates the conceptual foundations of ODM in terms of a set of key domain engineering challenges and ODM responses.
- **Part II: ODM Processes and Products**
  - *Section 4: Domain Engineering Life Cycle* — Describes the overall ODM process in terms of a domain engineering life cycle model.
  - *Section 5: Plan Domain Engineering* — Describes the ODM processes involved in planning a domain engineering project, including scoping and defining the domain of focus.
  - *Section 6: Model Domain* — Describes the ODM processes involved in modeling the commonality and variability of possible features in the domain.
  - *Section 7: Engineer Asset Base* — Describes the ODM processes involved in selecting the specific domain features to be supported in a reusable asset base and architecting and implementing the asset base.
- **Part III: Tailoring and Applying ODM**
  - *Section 8: Supporting Methods* — Describes several categories of methods and techniques (external but complementary to ODM) that can be applied to support aspects of the ODM process.
  - *Section 9: Optional Layers* — Describes several process layers that can optionally be added to ODM to provide capabilities that may be needed in some project contexts.
  - *Section 10: Guidelines for Applying ODM* — Provides a set of basic, practical guidelines for tailoring and applying ODM on a domain engineering project.

The guidebook also includes the following supplementary material:

- **Appendix A: ODM Process Model** — A graphical view of the ODM process model, expressed in the $IDEF_0$ notation. (The $IDEF_0$ diagrams are also embedded within the process descriptions in Part II.)
- **Appendix B: ODM Lexicon** — Definitions of the terms that are fundamental to understanding ODM.
- **Appendix C: ODM Workproducts** — Description of the ODM workproducts, with tem-

plates for selected workproducts.

- **References** — Bibliographic entries for all documents referenced in the guidebook.

## 1.5 How to Read the Guidebook

Each segment of the audience has different needs and interests and will thus benefit most from different portions of the guidebook. All segments of the audience should read Part I (in conjunction with the ODM lexicon in Appendix B) to gain a basic understanding of the method. The Program/Project Planner and the Process Engineer should read Part III to gain general insight into how the method can be tailored and applied. The Process Engineer and the Domain Engineer should read Part II and the appendices to learn about the ODM processes and work products in detail, but from differing perspectives: the Process Engineer to gain insight into tailoring and integrating the ODM process to support domain engineering, and the Domain Engineer to learn how the process can be enacted to produce reusable domain assets.

Although the guidebook can be read sequentially from beginning to end, it has been structured to support alternative reading styles. As indicated in the previous paragraph, Part I should be read first, but Parts II and III can be read more or less independently. Furthermore, the sections within Parts II and III need not be read in a strictly sequential order. This is due in part to the fact that the guidebook was developed in accordance with the Unisys STARS Process Definition Process, which imposes a clear discipline for structuring and presenting process information. The structure of Part II mirrors the hierarchical structure of the ODM process model, as depicted in graphical process trees and $IDEF_0$ diagrams. This structure makes it easy to find and read descriptions of specific portions of the process model. In addition, the sections in Part II have a well-defined internal structure that enables them to be read and understood relatively easily without a global understanding of the process. In practice, however, it is useful to read about a process in conjunction with the processes that surround it, including its parent processes. Further guidance for how to read and interpret the process descriptions is provided in an introductory portion of Part II.

Another important aspect of the guidebook structure is the treatment of key ODM terms and workproducts. In general, when a key term is first introduced in the guidebook, it appears in a ***bold italic*** typeface and is defined at that point. Such terms are also often shown in bold italic when first appearing within sections of the document to identify them as lexicon terms (this is not done consistently, however). All such terms also appear, with definitions, in the ODM lexicon in Appendix B so they can be easily looked up whenever they are encountered in the document.

The ODM workproducts and data items (all of which are represented as data flows in the $IDEF_0$ diagrams) appear in a SMALL CAPS typeface wherever they are referenced in the document. They are also included in the lexicon appendix, which is arranged in alphabetical order. However, the lexicon entries for the workproducts refer the reader to the full workproduct descriptions in Appendix C, which reflect the hierarchical workproduct structure. The description of each workproduct includes a reference to the process which creates it, thus providing an additional navigational aid to the reader. Appendix C also includes a set of templates for selected workproducts.

To further ease reading and navigation of the guidebook, the title and number (or letter) of the current section or appendix appears in the header atop each page.

NOTE: Because ODM is closely related to concepts in the STARS Conceptual Framework for Reuse Processes (CFRP) model, we recommend that the reader become familiar with the CFRP by reading either the CFRP Description document [CFRP93a] or a paper on the CFRP [Crep95a] prior to reading the remainder of the guidebook (particularly Parts II and III). See Section 2.5 and Section 4.0 for more details on ODM's relationship with the CFRP.

# 2.0 ODM Background

## 2.1 Origins

Organization Domain Modeling (ODM) is a systematic approach to domain engineering developed by Mark Simos of Organon Motives over the past seven years. The ODM approach originated during an early Unisys STARS project to develop the Reuse Library Framework (RLF), a hybrid semantic network and rule-based knowledge representation system intended for use as a domain modeling tool for software reuse [Unis88a, Sold89a]. (Mark Simos was initial technical lead on this project while with Unisys.) The RLF was developed by conducting a domain analysis of knowledge representation systems used in the expert systems field and producing a layered architecture and implementation based on the results of the analysis. The domain analysis approach developed on this project included many of the core ODM ideas and established an initial foundation for the ODM method.

Subsequently, considerable support and collaboration in refining ODM has come from the Hewlett-Packard Company and from Unisys Corporation, as a STARS prime contractor. In particular, the STARS Program has devoted significant resources over the past four years to refining ODM and defining its relationships with broader reuse process models and supporting tools. This is now culminating in the Unisys STARS Reuse Whole Product, in which ODM plays a central role (see Section 2.5 below).

The scope of the method has been further broadened through reuse consulting work with various organizations [Simo91b] and by synthesizing other research in domain analysis and domain engineering [Prie91a, ADER95a] with work in non-software engineering disciplines such as organization redesign and workplace ethnography [e.g., Seng90]. As a result, ODM emphasizes the organizational and non-technical issues associated with domain engineering more than comparable methods, while retaining a strong technical foundation rooted in a variety of technical disciplines.

## 2.2 Motivation

ODM was developed to address a gap in the domain engineering methods and processes available to reuse practitioners. Because work in domain analysis for software reuse emerged out of the software engineering research community, many of the methods were developed by technologists. Domain analysis was seen as primarily a technical modeling problem, often little more than a set of extensions to system modeling techniques to accommodate representations of variability. The resulting body of techniques range from informal to formal in terms of representations but are often quite ad hoc in terms of actual processes followed. More importantly, they leave unaddressed some of the most difficult problems would-be domain engineers face in trying to launch real pilot projects in business settings.

On the other hand, there has been a great deal of work done on the non-technical aspects of reuse adoption, including economic cost models for reuse, technology transfer strategies, and broad organizational models for structuring reuse groups within software engineering organizations. The problem with this work as a starting point for domain engineering is that it generally assumes massive changes on the part of organizations; these depend in turn on a high level of commitment from both top-level management and from the engineering trenches. In most practical settings, such commitment comes only after some pilot projects have demonstrated the viability of a reuse-base approach within that organization.

People trying to launch small-scale domain engineering projects have to negotiate the full range of technical and non-technical issues in planning, managing, and transitioning their work. Yet they typically will not have the support of an already existing reuse infrastructure within the organization. In fact, motivating organizations to adopt reuse practices on a more global scale may well depend on successful results from initial pilot projects. There is a pressing need for more detailed guidance on how to define and manage these initial domain engineering pilot projects. The ODM method is an attempt to meet this need and establish a foundation for evolving initial pilot projects into thriving reuse programs.

## 2.3 Objectives

Specific objectives of ODM include:

1) Make the domain engineering process more systematic, formal, manageable and repeatable. In particular, document where key boundary negotiation and scoping activities take place, to avoid the phenomenon of ad hoc designs being imposed as domain models.

2) Ground domain engineering projects in a specific organization context. Support a view of modeling activities as not only describing the state of the organization, but also directly contributing to its evolution.

3) Maximize use of legacy artifacts and knowledge as:

   — an empirical basis for systematic scoping of domain definitions, models and asset bases.

   — a source of domain knowledge.

   — potential resources to use in reengineering.

4) Avoid embedding hidden constraints in legacy systems and artifacts into domain models and assets.

5) Encourage exploration of maximum variability within the domain, including opportunities in various dimensions:

   — Reuse of artifacts from across the software life cycle.

   — Reuse of process as well as product assets.

   — Reuse in ways that facilitate use of static components and generative techniques, as well as hybrid strategies for asset implementation.

   — Reuse via definition of flexible, highly developed architectures for the asset base.

6) Provide effective strategies for selecting an intended scope of applicability for asset bases that is strategically appropriate for the performing organizations. In particular, apply techniques that limit the impact of combinatorial sets of choices in selecting range of variability.

7) Support evolution of the asset base, and the scale-up of the technology to support new kinds of organizations, organized around domains rather than around systems or products in a conventional sense.

Many of these objectives are mutually supportive. For example, systematizing the process (Objective 1) is closely related to effective scoping strategies (Objective 6); however, there are additional elements involved in systematizing the process; and systematizing the process is not the only reason for scoping strategies.

## 2.4 Scope and Applicability

ODM is intended as a highly tailorable and configurable domain engineering process model, useful for diverse organizations and domains, and amenable to integration with a variety of software engineering processes, methods, and implementation technologies. In terms of the STARS Conceptual Framework for Reuse Processes (CFRP) [CFRP93a], the core ODM method specifically addresses processes involving the creation of reusable assets, as well as elements of reuse planning and infrastructure development. ODM is thus directly applicable to planning and performing domain engineering projects in the context of an overall reuse program. (See Section 2.5 below and Section 4.0 for more information about ODM's relationship to the CFRP.)

### Where ODM Can Be Applied

ODM was developed primarily to support domain engineering projects for domains which are:

- mature (i.e., a set of legacy systems exists)
- reasonably stable (i.e., at least some of the legacy systems are worth examining)
- economically viable (i.e., new systems are anticipated in the domain).

ODM may be applicable in circumstances where not all these criteria are satisfied, but ODM is most applicable when they are all met. Also, since ODM has been developed in collaboration with both government and commercial software organizations, it is applicable to organizations and domains in either of these contexts.

ODM can be applied on domain engineering projects of arbitrary scope, in the context of reuse programs that are just starting or are very mature. It is recommended that organizations just beginning a reuse program apply ODM on a pilot domain engineering project focusing on a relatively small domain. This initial foundation can then be evolved into a broader reuse program.

### Where ODM May Not Be Applicable

In terms of the CFRP, the core ODM method does not directly address the ongoing management of domain models and assets, nor utilization of the assets by application development projects. ODM also does not encompass overall reuse program planning, including the establishment of producer-consumer relationships between domain engineering projects and other efforts, such as system reengineering projects or planned new products.

ODM may not be applicable within organizations that are not prepared to commit to, or at least experiment with, the notions of systematic reuse inherent in the CFRP and ODM. Also, not all organizations may be prepared to adopt the level of modeling rigor or the modeling styles or approaches recommended within ODM. Although ODM has been designed for integration with a wide range of supporting methods and tools, some organizations may not currently possess a technology infrastructure and level of technical expertise sufficient to support ODM modeling needs.

## 2.5 Relationship to Other Products

Within the STARS program, ODM is being developed and refined as part of the Unisys STARS *Reuse Whole Product.* The Reuse Whole Product includes a set of reuse support technologies that the Unisys STARS team has developed, integrated, or used. As part of the Reuse Whole Product effort, these technologies are being further integrated and augmented with a comprehensive set of training materials and examples to unify the technologies and make them easier for practitioners to use in concert. The Reuse Whole Product concept is inspired by Geoffrey Moore's "Crossing the Chasm" model of technology transition [Moor91a].

The Reuse Whole Product component technologies can be viewed as addressing reuse at differing levels, or layers, of abstraction. These layers, from highest to lowest level of abstraction, are termed *Concepts*, *Processes*, *Methods*, and *Tools*. In general, technology choices made at any layer will constrain the choices available at the layers below. The Reuse Whole Product reflects a specific set of technology choices made at each layer, as shown in Exhibit 1.

| Level of Abstraction | Products |
|---|---|
| Concepts | STARS Vision<br>Organon Vision<br>CFRP |
| Processes | CFRP<br>ROSE |
| Methods | ODM &<br>Supporting Methods |
| Tools | RLF<br>KAPTUR<br>ReEngineer |

**Exhibit 1.** STARS Reuse Whole Product Technology Layers

The technical concepts framing the Reuse Whole Product stem from the STARS vision of *mega-programming*, which integrates the concepts of process-driven, domain-specific reuse-based software engineering, supported by modern tools and environments [Boeh92a, Fore92a]. This vision is augmented in the Reuse Whole Product by the ***organon*** concept, which is founded on the notion of repositories of codified domain knowledge, coupled with proactive technologies to support the use and evolution of the knowledge [Simo91a].

Within this context, the Reuse Whole Product is scoped specifically to provide integrated process and tool support for the ODM domain engineering life cycle. In addition to ODM, the major technologies in the Reuse Whole Product are:

- *STARS Conceptual Framework for Reuse Processes (CFRP)* — The CFRP is a consensus STARS product that provides a conceptual foundation and framework for understanding domain-specific reuse in terms of the processes involved. [CFRP93a, CFRP93b, Crep95a]
- *Reuse-Oriented Software Evolution (ROSE)* process model — A CFRP-based life cycle process model that partitions software development into Domain Engineering, Asset Management, and Application Engineering and emphasizes the role of reuse in software evolution. [ROSE93a]

- *Reuse Library Framework (RLF)* domain modeling toolset — A toolset developed by Unisys and STARS which supports taxonomic domain modeling via semantic network and rule-based formalisms, and features graphical and outline-based model browsers. [Sold89a]
- *Capture* domain modeling and legacy management toolset — A toolset developed by CTA and NASA (and now being commercialized by CTA) which graphically supports comparative modeling of system artifacts and domain assets. [Bail92a]
- *ReEngineer* reengineering toolset — A toolset developed by Unisys to support the reengineering of legacy systems via fine-grained analysis and abstraction of system structure.

The Reuse Whole Product effort is underway and will continue through early 1996. Several evolving products and supporting materials will be released and available for trial use during that period (including this guidebook and a planned revision).

Although the above technology choices are sound in the Reuse Whole Product context, ODM can be applied in conjunction with a wide variety of other methods and tools that support the domain engineering life cycle. In general, this guidebook has been written to be as independent of specific support technology choices as possible. Although a catalog of potential ODM support technologies is beyond the scope of this guidebook, Section 8 describes several categories of methods (and related tools) that can support ODM. Please consult that section for more details.

## 2.6 Applications To Date

ODM has been applied on a small scale by a variety of organizations, including Unisys Corporation, the Air Force Comprehensive Approach to Reusable Defense Software (CARDS) program, and the Software Engineering Institute. In addition, the following relatively major ODM application efforts are underway and have produced good results:

- **Hewlett-Packard.** Some aspects of the ODM approach were evolved in working with HP application teams in RADAR (Reusability Analysis/Domain Analysis Review) sessions [Coll91a]. These were design reviews focusing on projects that have followed a system design or architectural modeling process similar to domain analysis, whether or not a formal domain analysis method, per se, was followed.

  The ODM process model was used as a basis for question sets use in the sessions to probe engineering teams to reflect both about the products and processes of domain analysis. The sessions served both as training for engineers in the concepts and methods of domain modeling, and to rapidly reveal the gaps and breakdowns in the method. Patricia Collins (HP Corporate Engineering) has since evolved these question sets into a domain engineering workbook specifically tailored to HP organizational objectives and business culture. This workbook is now being applied on several small-scale domain engineering efforts within HP divisions.

- **Army STARS Demonstration Project**. STARS is working with the Army, Navy, and Air Force to sponsor three DoD software engineering projects (termed the STARS "Demonstration Projects") to assess the megaprogramming paradigm in realistic and familiar contexts. Unisys is supporting the Army STARS Demonstration Project, which is being performed by the US Army Communications and Electronics Command (CECOM) Software Engineering Directorate. The project focus is on domain engineering and system reengineering in a system maintenance context.

  ODM, in conjunction with other Reuse Whole Product technologies (particularly the CFRP and RLF), has formed the basis for the domain engineering approach that has been developed and applied on the project [ADER95a, Wick94a]. The experiences and feedback from adapting ODM for use on the project have substantially influenced the evolution of ODM into its current form. In addition, the project's experiences in applying RLF and other domain engineering support technologies have shed considerable light on ODM tool support strategies in general. The project has significantly formalized its domain engineering process and is developing a Domain Engineering Guidebook [DEGB95a] that describes the process that will be recommended for use on future projects.

# 3.0 ODM Core Concepts

Since the seminal work of Neighbors and other researchers in the early 1980's, domain analysis (DA) has emerged as a distinct area of research and practice within the software reuse research community [McNi88a, Neig83a]. There are now a number of published approaches to DA, including a range of informal and formal methods and processes [DISA93a, Goma90a, Kang90a, Prie91a]. More recently researchers and practitioners have attempted systematic comparisons of DA methods and technologies [Prie91b, Wart92a]. Rather than searching for a single best-practice DA method, such comparative work generally aims to assist practitioners in selecting DA method(s) appropriate for their organizations and domains. To date, however, no clear consensus has emerged on key differentiators for DA methods. This lack of consensus reflects underlying disagreements about definitions of DA.

DA has arisen at the confluence of several overlapping system engineering disciplines and methods, and DA methods embed different degrees of commitments to these related technologies. For example, many DA methods are strongly coupled to particular preferred techniques for system modeling, analysis and design, e.g., use of Ada design principles [McNi88a] or object-oriented approaches [DISA93a, Goma90a]. Comparisons of DA methods, therefore, are easily sidetracked into focusing on such interdependencies.

To clarify these issues and establish a frame of reference for understanding ODM core concepts, Section 3.1 defines domain engineering concepts in the context of overall software engineering needs and approaches. Section 3.2 then describes key characteristics of the ODM domain engineering life cycle in terms of these basic concepts, and Section 3.3 explores a set of key domain engineering challenges and explains how ODM responds to them.

## 3.1 Domain Engineering in Context: Definitions

Domain engineering is conventionally associated with a focus on multiple rather than single systems. However, domain engineering in its full generality involves more than this distinction. Some terminology is introduced below to distinguish various engineering contexts, as illustrated in Exhibit 2. (These terms are simplifications of a complex spectrum of concepts and are not intended as any kind of standard.)

***(Forward) system engineering*** addresses the problem of implementing a new system for a single application context[1]. ***System reengineering*** addresses the task of restructuring and/or reimplementing some or all of an existing legacy system. The term ***variability engineering*** is introduced here to suggest application and extension of systems engineering techniques to multiple-system contexts, such as product-line planning and engineering. This would encompass the representation of variability both within and across systems (via ***variability modeling*** techniques) as well as various design and implementation techniques to cost-effectively support variability, via flexible architectures, generative technologies, etc.

By analogously extending reengineering techniques to multiple legacy systems, we can speak of ***variability reengineering*** techniques as well. Note that one desired outcome of good (forward)

[1] The term ***system*** is used here to denote an application designed for a single context of use. There is no intent to imply a *systems* (i.e., hardware, software, practitioners) vs. *software*-only distinction. The extent to which DA techniques are software-specific is a complex issue beyond the scope of the guidebook. What constitutes a *single application context* will vary from organization to organization. However, almost by definition, a domain will represent system functionality that spans multiple contexts in the organization's own terms.

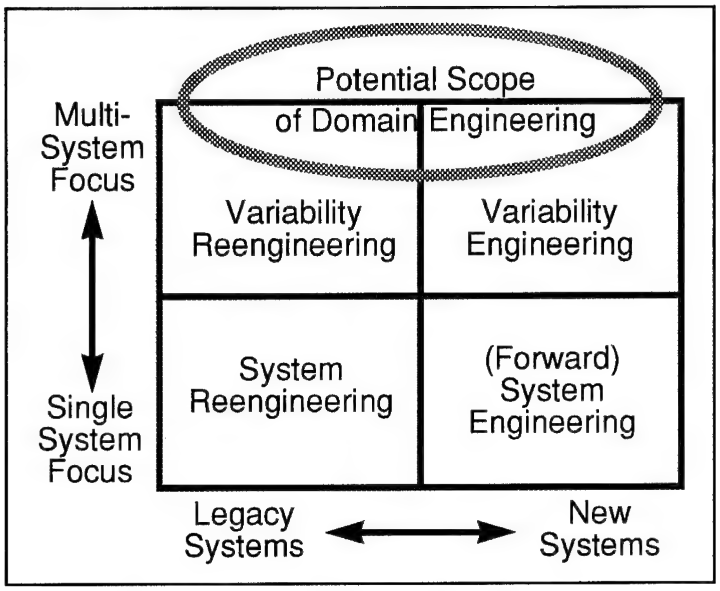


**Exhibit 2.** Domain Engineering in Context

variability engineering would be divergence, or ability to create flexible and adaptable systems; in variability reengineering, one desired outcome would be convergence, or consolidation of the set of legacy systems to be maintained.

A software ***domain*** is defined here as an abstraction that groups a set of software systems or functional areas within systems according to a ***domain definition*** shared by a community of ***stakeholders***. A domain can be represented by a ***domain model***, which serves as a basis for engineering components (or ***assets***) intended for use in multiple applications within the domain. The domain model provides a formalized language for specifying the intended range of applicability of the assets. The term ***domain modeling*** denotes development of the domain model; ***asset base engineering*** denotes architecting of the asset base and implementation of assets; and ***domain engineering*** represents the entire life cycle.

Domain engineering thus addresses problems involving multiple new and/or legacy systems, interacting in potentially complex ways. For example, multiple legacy systems may be available for reverse engineering in parallel, and reengineering those systems may result in assets that can be used in multiple applications; such usage could involve back-filling reusable assets into legacy systems as well using the assets to build new systems. Domain engineering must focus on variability engineering/reengineering techniques to manage the complexities of engineering across multiple system contexts. This is a central theme of the ODM domain engineering approach.

## 3.2 The Core ODM Domain Engineering Life Cycle

ODM is structured in terms of a core ***domain engineering life cycle***, distinct from and orthogonal to the system engineering life cycle. This life cycle focuses on:

- Selecting, scoping, and defining target domains
- Modeling the range of variability that can exist within the scope of the domain
- Engineering an asset base that satisfies some subset of the domain variability, based on the needs of specific target customers

The processes and work products of this core life cycle specifically address the domain engineering issues cited in the previous section; all other activities are allocated to ***supporting methods***.

The core life cycle is represented by a process and workproduct model that can be tailored and instantiated in a variety of ways. Since each organization, and to some extent each domain, will have unique constraints and preferences, the goal is a core domain engineering method that can be integrated with a broad variety of supporting methods. Supporting methods include system modeling techniques and taxonomic modeling techniques (among other methods). In addition to supporting methods, invoked at particular points in domain modeling, the core ODM process model can be extended with optional ***layers*** which have pervasive impact across the entire life cycle.

Because activities classified as supporting methods are considered to be external to the core ODM process model, ODM does not directly address some steps conventionally considered part of domain engineering. For example, some methods recommend development of a domain requirements model and a domain architecture/design model. These respectively model commonality and variability within the "problem space" and a (high-level) "solution space" for the domain. These may be useful ways of partitioning or sequencing models in domain engineering, but the distinction reflects a *system* engineering life cycle (requirements — design —implementation), whereas the core *domain* engineering life cycle cuts across the system life cycle phases and can apply to any or all of them simultaneously. In some situations, a domain model of the requirements for a system might be useful as a stand-alone domain engineering result. Note that the development of domain models reflecting the system life cycle, such as those mentioned above, could occur within the ODM life cycle, but would result from selecting and applying a particular set of supporting methods.

The fact that domain engineers are creating asset bases rather than conventional systems introduces considerable engineering complexity. The core ODM life cycle is oriented towards ***architecture-centric domain-specific*** reuse. This approach assumes that multiple assets will be used in concert in particular applications. Design of individual assets may be influenced by their anticipated placement within the asset base; thus the asset base itself must be architected and designed in a coordinated way.

## 3.3 Key Challenges and ODM Responses

Given the nature of the domain engineering problem ODM addresses, the primary goal of the method is the systematic transformation of artifacts from multiple legacy systems into assets that can be used in multiple systems within an architecture-centric domain-specific context. The intent is to:

1) provide a systematic basis for reengineering assets for reuse, based on explicit documentation of their intended scope of applicability; and

2) maximize use of existing legacy artifacts as a basis for both domain knowledge and reengineering.

Naturally, any successful domain engineering project will need to apply sound software engineering techniques in achieving these goals. In addition, as discussed above, a repertoire of techniques for modeling and engineering variability will be essential to achieving economies of scale and reuse that will make domain engineering feasible. However, there are a number of key methodological challenges inherent to domain engineering that are not simply addressed by either conventional software engineering techniques or expanded capabilities for accommodating variability. These challenges, and the ways in which ODM responds to them, are explored below in detail. They include:

- Reuse in the legacy system context
- Implicit domain boundaries
- Dynamic problem and solution spaces
- Transforming artifacts into assets
- Architectural variability

## CHALLENGE: Reuse in the Legacy System Context

There are complementary risks to be considered in domain engineering. Independence from architectural and design decisions of specific legacy systems is critical, as these may not scale up to be applicable across the entire range of systems desired for the domain. On the other hand, in mature domains obtaining maximum value from legacy systems is essential. Otherwise domain engineering amounts to system design with variability, dependent on the architecture and design expertise of asset base engineers who may not know the rationale behind the design of existing systems. Since domain assets, if accepted, will be applied across many systems the potential impact of poor design is much broader than if made within a single system development effort.

## ODM Response: Descriptive-Prescriptive Separation

In part to address these dual dimensions of risk, the ODM process model separates the life cycle into two distinct phases: the *descriptive* domain modeling phase, and a *prescriptive* asset base engineering phase. ***Descriptive modeling*** documents what experts have learned about how to build particular classes of applications through the experience of developing multiple systems. In descriptive modeling, knowledge about the domain is obtained in part by analyzing legacy systems through the intermediary definition of domain ***features***, significant differentiating capabilities across domain systems. Modelers document commonality and variability in system structure and function and attempt to recapture the rationale for decisions embedded in those systems.

The asset base engineering part of the life cycle marks a shift to ***prescriptive modeling***, where binding decisions and commitments about the scope, architecture and implementation of the asset base are made. The descriptive feature model is transformed into a prescriptive set of committed features for implementation. These prescriptive features are mapped onto the structure of the asset base and to sets of specifications for particular assets. By preserving traceability from features back to exemplar artifacts, asset developers can gain access to potential prototypes on which to base development of new assets. In addition, because assets are described within the asset base in terms of the domain feature model, asset utilizers can eventually retrieve components based on the same descriptive feature language.

Descriptive domain modeling is somewhat analogous to ***reverse engineering*** in a reengineering context; or to the "current physical" and "current logical" models (sometimes called the "as is" model) in Yourdon systems analysis techniques. However, there are significant distinctions as well. In ODM descriptive models can contain information about requirements for new and anticipated systems as well as legacy systems. Conversely, customer systems for the asset base might include legacy systems to be reengineered. The descriptive model is best thought of as a language for expressing a *coherent space of possibilities* within the domain. This model can include aspects of *both* the problem and solution space, as well as interpretive rationale connecting them. The key principle of the descriptive phase is that no binding decisions or commitments are made about the functionality to be supported in the asset base.

## CHALLENGE: Implicit Domain Boundaries

The emphasis on a multiple-systems scope suggests another primary methodological challenge in domain engineering: that of domain boundary definition and scoping. In system engineering the problem scope is to some extent a given. The single context of use establishes a scope within which requirements can be defined. In domain engineering there is no single system context to establish clear entry and exit criteria for modeling tasks. Where does the scope in a multiple-system effort come from? How many systems are enough, or too much? These issues are compounded when potential scope can include legacy systems that are candidates, both for information-gathering and for potential reengineering using domain assets.

In fact, these issues apply in any engineering context, where designers' scoping decisions are usually implicit, informal and private. Being more explicit, systematic and public about how scoping decisions are made would benefit any software engineering project. However, systematic scoping is critical to the very feasibility of domain engineering.

## ODM Response: Explicit Scoping

To make scoping explicit in domain modeling, the ODM life cycle is structured as a series of incremental scoping steps. Each step builds on and validates previous scoping decisions and offers different opportunities for refinement and focusing. This aspect of the ODM life cycle is illustrated in Exhibit 3.

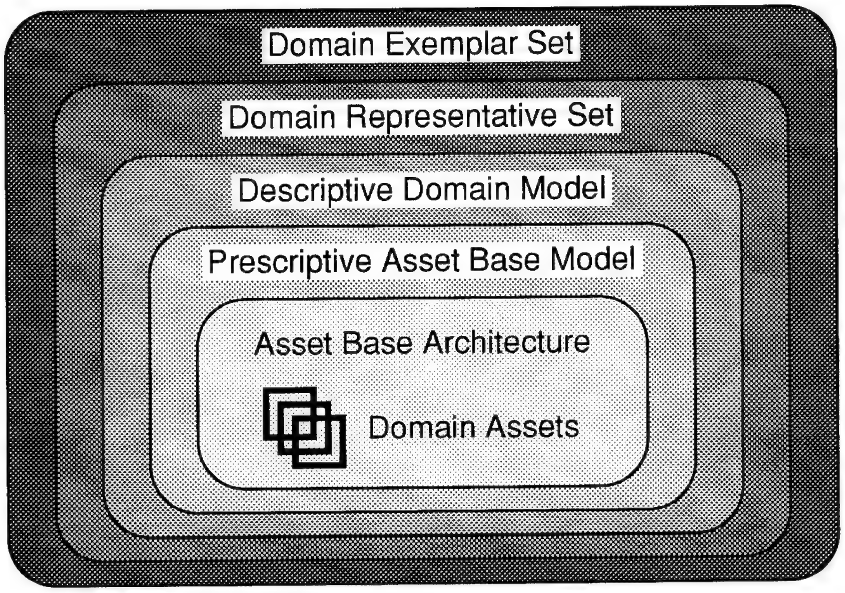


**Exhibit 3.** Iterative Scoping Steps in ODM

The domain definition process in ODM is formalized by specifying rules of inclusion and exclusion bounding what is in and out of the domain. This ***intensional*** domain definition is empirically grounded and validated with reference to an ***extensional*** definition: a set of example systems classified in terms of domain membership (i.e., ***exemplars***, ***counter-exemplars*** and ***borderline exemplars***). The domain is also ***oriented*** with respect to various axes: historical (e.g., predecessor or anticipated successor technologies), operational (e.g., subsystems or applications within which domain functions are embedded), and conceptual (e.g., broader domains, more specialized domains, analogy domains). The resulting ***domain interconnection model*** includes formal representation of domain-to-domain relationships with respect to the selected domain (or ***domain of focus***).

Defining and scoping the domain is distinct from scoping the data that will be acquired to model the domain. A subset of exemplar systems is selected as a ***representative systems*** set for intensive analysis. Further scoping activities are performed to select the best cross-section of data to gather from various system life cycle phases of these representative systems, and which specific individual models will be developed from the data. The resulting set of descriptive models is formally integrated and interpreted, establishing rationale and contextual information about variability within the domain.

The final domain model is extended beyond a purely descriptive model through innovative transformations applied directly to the model. This extension step is necessary to ensure that the prescriptive asset base model is always a subset of the domain model.

The prescriptive phase begins by re-scoping the range of functionality to be supported by the reusable assets to be developed; commitments are made to a real set of customers for the asset base. These commitments cannot be finalized in the initial project planning step, because they depend on data that emerge from the descriptive modeling phase. This scope is documented in the ***asset base model*** which therefore represents a subsetting of the final, extended domain model. This prescriptive model leads to the development of a ***asset base architecture*** to support the prescribed variants. After this architecture has been specified, asset implementors may still choose to develop only a subset of the assets specified in the architecture context.

Successive levels of scoping in ODM should result in a narrowing of the domain engineering problem into specifications that can be implemented using the appropriate supporting methods of system engineering, augmented with variability engineering techniques.

Throughout the life cycle, scoping decisions serve as a risk mitigation strategy, constraining and managing the process. At the same time, design and modeling decisions made at each point have a rich foundation of data on which to draw because of previous steps. These techniques reflect a fundamental principles in the ODM approach to domain modeling: the reusability of an asset can only be assessed relative to explicit documentation of its intended scope of applicability.

## CHALLENGE: Dynamic Problem and Solution Spaces

Explicit and iterative scoping techniques in ODM serve to make the boundary decisions in domain modeling explicit and public. They also serve to isolate *definitional* processes from *modeling* processes, which can mitigate some of the breakdowns typical in implicitly scoped design situations. However, they do not provide direct guidance in *how* to select and scope the domain and asset base.

Problems of selecting scope are less likely to become visible when the development effort is guided by a clearly defined problem space (e.g., user/customer/market requirements) and/or a clearly defined technology solution space (e.g., architecture, technology availability, development philosophy). In a problem-driven engineering context engineers seek solutions with respect to a somewhat stable problem. In a product- or technology-driven context there is a stable, if evolving, solution space for which engineers seek problems, or rather opportunities for application. But typically in domain engineering contexts both the problem and solution spaces are simultaneously dynamic.

There are many simple examples outside of the software engineering context for these kinds of situations, e.g., simultaneous arguments over who will be a member of an organization and the proper mission of the organization. Certain breakdowns are symptomatic of these situations, such as a tendency for boundaries to "wander", or for circular conflicts that alternate between discussions about definitions and decisions, etc. In domain engineering, these process breakdowns can have major repercussions on architectures for entire system families and product lines.

## ODM Response: Contextual Methods

ODM addresses this issue with the recognition and assertion that domains are always socially situated or socially constructed; that is, they are always shaped by the context of multiple overlapping communities of use, development, maintenance, customization, and application. As implied in the name of the method, ODM deals not with "domains" in the abstract but with ***organization domains***[2]. ODM uses explicit modeling of social and organizational context to ground the entire modeling life cycle, from articulations of domain engineering objectives and domain selection criteria, domain identification, definition and scoping, through domain modeling and up to and including selection of technologies for asset implementation. Organizational context also establishes entry and exit criteria for modeling activities otherwise quite difficult to bound.

Thus, domain planning in ODM begins by modeling the set of ***stakeholders*** for the domain engineering project: specific groups and/or individuals with interests and objectives relative to the project and the domain, as well as diverse viewpoints, experience and terminology. Documentation of stakeholder context grounds domain selection by formally modeling the strategic business setting in which the domain engineering project is being performed. Key stakeholder roles include domain ***informants*** and potential ***customers*** for reusable assets; other roles, such as project sponsors, domain engineering technology providers and the modelers themselves are also part of the model.

Though there are multiple stakeholders in any system development effort, domain knowledge is typically distributed, not only across multiple systems, but across organizational, departmental, project and product line boundaries. This creates complex trade-offs where requirements may differ in priority or even be inconsistent, and even basic domain terminology may be interpreted in diverse ways. Domain selection and even later modeling decisions often involve boundary negotiations between diverse stakeholder interests. Obviously, the point of context-setting activities in ODM is not to re-invent the art of stakeholder analysis and conflict resolution; as with the scoping steps discussed previously, the aim is to reduce contextual dynamics to forms amenable to supporting methods and techniques.

**Contextualizing Domain Data.** Documentation of context is also important throughout the modeling process. The language, values, assumptions and history of each relevant community introduces contextual information into software ***artifacts*** that invisibly constrain those artifacts. This "hidden context" in software artifacts is one of the primary sources of uncertainty in ad hoc reuse of legacy components. Because many constraining assumptions come from the cultural context in which software artifacts are developed and used, the domain modeling process must identify and make explicit how this kind of information is embedded within the artifacts.

Unearthing this implicit context, fundamental to overall aims of domain engineering, requires a rich set of data acquisition techniques. Most DA methods recognize the need to gather information from both artifacts and interviews with domain experts. The ODM approach to data gathering

[2]. The term "organization" here is a place holder for various entities of the organizational context in which domain engineering is performed. This could be a department or division within a larger organization, a group of organizations linked as suppliers, value-added resellers and customers, a market segment, or a community with shared interest in some specific technical area.

reflects the principle that a domain model can only be validated relative to explicit documentation of the information sources from which it was derived.

Several complementary techniques can be used, including: examination of legacy artifacts; structured interviews with ***informants*** (including not just domain "experts", but system users and other practitioners in the domain); and ***ethnographic techniques*** such as direct observation or participant observation of task work flow in the domain. Use of complementary data gathering techniques creates robust opportunities for cross-validation. Strategic use of multiple representative systems is also integral to obtaining the benefit of certain cognitive insights that are elicited through perceived variation across similar systems.

This cognitive effect means that data acquisition must be treated as an active intervention in the stakeholder communities for the domain. Especially in group interviewing settings, informants may interact with people in the organization (or from other organizations) that they would never have met in an ordinary project-centered context. This can result in significant learnings about the domain, for modelers and informants.

**Contextualizing the Domain Language.** In many ways, the goal of the domain modeling phase is to specify a domain-specific language rich enough to represent specifications for either individual applications or reusable assets for the domain. Development of a domain ***lexicon*** serves as the transition from purely descriptive data acquisition to comparative modeling. The domain lexicon provides a terminological map that captures and centralizes the specific language of the domain. Since domains often cross organizational boundaries, differences in domain informants' use of terms must be reconciled, as well as differences between informants' and modelers' terminology. Within the context of the formal modeling techniques employed, the lexicon can be seen as the "lexical analyzer" for the domain language whose semantics are specified in the formal domain models.

At the conclusion of formal descriptive modeling, a further contextualizing step takes place in interpretive modeling. Here, additional contextual information is elicited to determine the rationale for commonality and variability observed in the representative systems set.

## CHALLENGE: Transforming Artifacts into Assets

The techniques outlined above—separating descriptive from prescriptive modeling phases; explicit and iterative scoping; context-setting activities in domain selection and information gathering—form a strong systematic basis for domain engineering. However, in order to support the goals of architecture-centric reengineering of artifacts into assets a further degree of formalization in domain models is required. Certain techniques from library science have been adapted to classification schemes for domain models [Prie91a]. But the analogy to conventional libraries breaks down for domain engineering when viewed as a parallel reengineering of artifacts to assets. The correct analogy would be a library classification scheme that supported the potential re-writing and restructuring of all books in the library as part of the acquisition process. Clearly these classification techniques were not designed for this purpose. Some requirements for formal modeling support are discussed below.

**Process Assets**. Contextual analysis of domain practitioners allows for encapsulating domain *processes* as well as *products* as reusable assets. For example, a checklist of completeness and consistency criteria for domain requirements may be reused as an asset by requirements analysts, though never incorporated verbatim as a "component" of an application's requirement specifications. However, this requires techniques to model developer as well as operational contexts for domain systems.

**Generative Implementations.** Closely connected to the integrated process and product view of reuse is the goal of supporting generative as well as component-based implementations of domain functionality. Generative solutions can be appropriate, for example, in situations where continuous variation or combinatorial combinations of feature variants make reuse via a collection of static components impractical. Since the final choice of what features to implement is made in asset base modeling, then ideally domain modeling methods should defer commitment to component vs. generator-based implementations to these later stages of asset base engineering. Such models would also prove more robust as the domain asset bases evolve with usage; for example, if a collection of static components is replaced with a generative implementation.

**Feature Binding Time**. Since domain models describe the commonality and variability across multiple systems, they must distinguish those combinations of features that must be *simultaneously satisfied* by a single system from combinations where each feature *individually* must be supported by some system derivable from the domain model. Essentially, the notion is that a choice between variant features can be "bound" or committed at various points within the software engineering life cycle. A simple example would be the choice between tiled and overlapping window protocols in a window management system. Some systems hard-wire specific choices of tiled or overlapping windows. Others allow this to be specified in a configuration file read at start-up time. Still other systems might have a "preferences" switch that can be changed dynamically at run time by the user.

It is tempting to assume that the latest (i.e., most "flexible") feature binding time is inherently the most "reusable", since in theory such a system can simulate the behavior of system variants with earlier binding times. However, this ignores the many design and performance trade-offs that may be involved in selecting the binding time for a given feature. For example, there may be systems where it is specifically *not* desirable for users to be able to switch from tiled to overlapping windows.

Conventional system modeling representations (even object-oriented representations) may have inadequate means for distinguishing a system version that binds a choice at implementation time vs. at run time. A run-time operation in one system may appear as a design decision in another. At a minimum, representations of commonality and variability should have sufficient semantic expressiveness to distinguish these cases, and the domain modeling process must be able to utilize these representations effectively.

## ODM Response: Formalization of Features

In order to support these requirements, ODM integrates the scoping, contextualizing, and descriptive modeling activities described earlier with more formal modeling processes and work products. This involves use of ***formal features*** as an integrating mechanism throughout the life cycle. Features are defined empirically, tracing back to representative systems and artifacts. They are also defined contextually, being associated with particular domain practitioners (e.g., user-visible features, developer-oriented features, design factors) and domain contexts.

Formal modeling of features generally supports the extension of the domain model in the transition from descriptive to prescriptive modeling. To accomplish this, however, it is necessary that features be formally defined as entities themselves. To the extent that features can be detached from descriptions of representative systems and artifacts, and from specifications for assets, it becomes possible to do a kind of "second-order" feature modeling wherein feature models can be designed and transformed directly. ODM supports this approach by layering features atop ***concept models*** that provide fine-grained semantic interpretations of features.

In the interpretive modeling process, modelers document connections between the same logical feature as it occurs at different ***feature binding sites***. In the final, innovative transformation phase of domain modeling, feature variants can be shifted within and across contexts to different binding sites. For example, a simulation environment used by integration test engineers might be used by system operators to capture scripts and macros to build new reusable assets codifying their specific work practices. Opportunities for generative reuse are often revealed by such contextual shifts.

This is one example of a repertoire of model transformation techniques defined within ODM that can operate directly on concept and feature models. The goal of these formal modeling techniques, still an area of active research, is more than just exploration of innovative possibilities for novel features and feature combinations, although this is a valuable side-benefit. The real purpose of extending the domain model in this way is to achieve ***closure*** of the domain feature space. Without this step, the feature model would reflect the purely accidental descriptive boundaries of the representative set, or the pragmatic choices of features oriented towards anticipated customers of the asset base. Such a feature model would not necessarily prove robust and evolvable over time. The design goal in extending the domain model is therefore to discover the archetypal domain model approximated by the descriptive process.

## CHALLENGE: Architectural Variability

Each of the core concepts of ODM presented so far—separation of descriptive and prescriptive modeling, explicit and iterative scoping, contextual methods, and formal modeling techniques—has been shown to be a necessary element of a domain engineering method robust enough to address the problems of reengineering legacy artifacts into assets with predictable scope of applicability. These elements alone would produce a method able to address the construction of *individual assets*, implemented using a spectrum of compositional or generative techniques, and documented by a formal specification expressed in a feature language attuned to the domain-specific terminology and contextual interests of practitioners. This would represent a significant advance in relation to current informal methods of domain engineering.

However, it is clear that the process as described will require a non-trivial investment of resources in comparison to single-system engineering. To demonstrate sufficient economic benefit it is important to select domains with great care, and to ground projects in organizational contexts with clear incentives for long-term economies of scale and of reuse. The ***architecture-centric domain-specific*** view of reuse is based on the claim that such economies of reuse will be best achieved by building collections of assets, or ***asset bases***, focused around highly specific application areas, and organized into coherent architectures enabling large-scale reuse of aggregates of individual components. A central problem in this approach to reuse is the design trade-off between supporting large-grained reuse by application developers (which requires *structural integration* of components) and supporting *high variability* in structure to enable reuse within a broad market of applications.

Besides supporting the economics of reuse, an architectural approach to implementation of asset bases addresses a problem actually introduced by the increased formality in modeling described above. Fine-grained modeling of feature variants, choice of generative and component techniques, and the survey of diverse contexts for reuse all combine to create a potential combinatorial explosion in variability. More specifically, these techniques *expand modelers' abilities to envision potential variability* for their products, and thereby their need for techniques to deal with it effectively. Reducing variability through architectural "chunking" into manageable sets of features is one important technique.

In one sense, though, this architectural strategy for managing variability only promotes the problem to a higher level. Support for variability at the architectural level is one of the current areas of active research in the field of software architecture [Clem95a]. The final motivating theme in ODM is an attempt to systematize domain engineering in a way that facilitates the specification and implementation of ***highly variable architectures***. The following paragraphs provide some motivation for this goal.

As illustrated in Exhibit 4, the way domain boundaries are drawn with respect to their exemplar systems has a dramatic effect on the requirements for support of architectural variability within those domains. The grid represents a range of possibilities shaped by two dimensions. *Historical relatedness* refers to the "genealogical" relations among systems: whether the systems were developed, maintained, used by the same people in the same settings. *Structural similarity* refers to the topological closeness of systems, in terms of their component structure and interconnections.

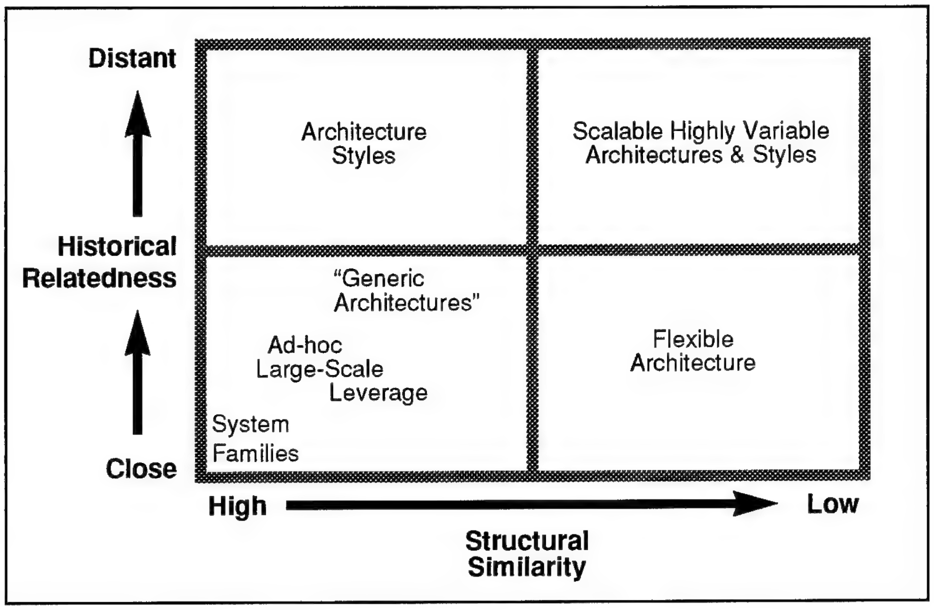


**Exhibit 4.** The Variability Barrier in Software Architecture

Although most examples that come readily to mind suggest close correlation between these dimensions, they are independent. System families in the conventional sense reflect lineages (as the term "family" implies), usually built by the same organizations (perhaps the same people) and often delivered over time to the same customer or market. But not all such lineages exhibit structural compatibility from system to system; this is a characteristic that varies from domain to domain. As we move away from system families with high structural similarity, we encounter the need for ***flexible architectures***. Since the current state of the art does not support the systematic construction of systems along these lines, tractability usually depends on ad hoc techniques and a high degree of expertise on the part of developers, supported by ***contextual knowledge***.

Conversely, researchers in software architectures have recognized dramatic structural similarities in the architectures of systems built to perform diverse real-world missions, and developed and used by diverse communities. Some of these structural patterns are beginning to be codified and catalogued as ***architecture styles*** [Shaw94a, Shaw95a].

As the diagram in Exhibit 4 suggests, current architectural practices can succeed to the degree that they can exploit either high structural similarity or contextual closeness, in system topology and genealogy respectively. But problem-space and solution-space structures do not directly correlate in general; and in many practical business settings, *problem domains a*re the motivating charter behind domain engineering initiatives.

This presents a formidable challenge for domain engineering methodology. At a minimum, it would be desirable to at least be able to identify and avoid domains where current architectural methods do not suffice. A more ambitious goal is to develop a repertoire of techniques that can address the "variability barrier:" scalable, highly variable architectures and architecture styles, explicitly embedding enough contextual information to support reuse in diverse application areas and organization contexts.

## ODM Response: Variability Strategies

ODM represents only a small step towards these goals. This document does not provide complete solutions for these long-ranging research problems. However, the core elements of the ODM process presented thus far in this section are all *necessary*, if not sufficient, to achieving reuse at this scale. In addition, the task structure in the latter phase of ODM, *Engineer Asset Base*, is intended to be scalable to asset bases architected to support high variability and diverse contexts of use. Advances in supporting methods in software architecture and related areas will be required to fully achieve these engineering objectives.

In the absence of complete supporting technologies, the following near-term strategies are offered for approaching the variability barrier in architecture:

- Pick small domains. Domains defined at the sub-system level have a better chance of scaling to highly variable architectural approaches. Here, "architectural" takes on a modified meaning: it refers to the external interfaces of such domains to their surrounding system context (both as invoked and as invoking services), and to their internal structure and topology.

  In general, the farthest toward the barrier, the smaller the domain scope needs to be drawn. So, for example, it would be feasible to address a large-scale family of applications with a domain engineering approach, but only if a relatively inflexible architecture (e.g., a standard architectural framework or, at most, a "generic" architecture with variable "plug-and-play" components) were specified. In many organizational settings, however, there are reasons why such an architectural approach will not be adopted. Tackling a large-scale domain in such an environment is therefore a high-risk strategy.

- Apply generative techniques at the architectural level. Generative techniques are a classic strategy for managing variability. When a subroutine has gained too many parameters with cross-constraints and dependencies, it can often be replaced by a generative solution that is much easier to use and can generate optimized solutions as well. In principle these same techniques can be applied to the configuration or "gluing" of large-scale components into aggregates. This approach will break down, however, if variability needs to be supported at multiple, interacting structural levels simultaneously. In this case, the problems of tractability at the variability barrier re-assert themselves.

- Prioritize specific customers carefully. Being clear about the organizational context that the architecture needs to support is essential to any systematic approach to the problem. Designate specific customers; but *more than one*. The added awareness of contextual dependencies and potential variability from building to support two or three customers is an order of magnitude greater than for a single customer; and far more feasible than building for "all customers" in general (but none in particular).

- Use a phased implementation strategy. Ignoring opportunities for variability will simply back developers into corners and limit possibilities for later evolution. Conversely, attempting to implement for too many dimensions of variability simultaneously will exceed the capacity of current techniques. An incremental strategy provides a middle ground—building assets of limited scope in the context of a long-term plan for evolving towards greater variability over time.

- Formalize to support greater variability. This includes formality at the representation level as well as increased process awareness and systematic methods. Where taxonomic and architectural representations are used, they will need to be more tightly integrated at potentially a fine-grained level, in order to address highly variable architectures. Alternatives and structural configurations need to be expressible at arbitrary points within the system structure.

  For example, the notion of a generic architecture as a high-level invariant topology with components that can be optional or variable really reflects the assumption that taxonomic modeling of a simple sort can be introduced uniformly at *one* structural level in the system; structural modeling is used above this level. This assumption will not hold in situations requiring highly variable architectures.

- Maximize variability in the descriptive phase of modeling. ODM encourages descriptive modelers to be quite brave in modeling variability. The suggestion to explicitly model genealogical relations between candidate representative systems, for example, acknowledges that domains can be defined to encompass systems from quite distinct lineages and organization contexts. Similarly, modelers are encouraged to choose representative systems that have diverse structural embodiments of domain functionality, e.g., an encapsulated subsystem in one context, distributed functionality in another.

  The rationale in this intentional seeking out of high diversity is to use the *descriptive* modeling phase to generate as much insight as possible about the potential range of variability. This would be a high-risk approach if there were not complete independence between the descriptive and prescriptive phases. Because asset base engineers can select whatever subset of the DOMAIN MODEL is strategically appropriate, they can scope the asset base in a variety of ways starting from a DOMAIN MODEL that is rich in variability. Separation of design processes into descriptive and prescriptive phases is therefore a recurring strategy at many levels throughout the ODM life cycle. Wherever it appears it is being used in part as a technique to regulate and manage variability.

## 3.4 Core Concepts Summary

Motivation for the ODM life cycle is presented above in terms of a set of key challenges faced in domain engineering and the responses to those challenges offered in ODM. ODM represents one approach to addressing these challenges. It is valuable to consider these issues when undertaking any domain engineering project.

Many unsolved problems remain. To transition the ODM method into widespread use, it is clear that tailored versions that provide fully integrated supporting methods will be needed. The degree to which the various supporting methods can be separated from each other and from the core life cycle remains an area of active research.

The current ODM method isolates these problems in a way that can be addressed by existing supporting methods and tools. Issues involving the integration of ODM with other technologies will be clarified as ODM is applied in practice and lessons are collected. ODM as it stands now is a useful starting point for practical domain engineering.

# Part II: ODM Processes and Products

This part of the guidebook describes the ODM process model and work products. This introductory portion describes how the part and its sections are organized, defines the conventions used to present information within the sections, and provides guidance in how to read and interpret the information presented.

In general, the process model is described in accordance with the notations and conventions associated with the Unisys STARS Process Definition Process [Klin95a].

## Part Structure

This part is organized hierarchically, reflecting the hierarchical structure of the ODM process model. The full model hierarchy is shown as a process tree in Exhibit 5 on the next page. Each node in this tree represents an ODM process. The nodes below a given process represent the subprocesses that are carried out in performing that process. The following terminology is used in referring to processes at each decomposition level within the process tree:

- ***Life cycle***: The top level, or "root" node, of the process tree (i.e., *Domain Engineering*), representing the overall ODM domain engineering life cycle.
- ***Phase***: One of the three major components of the ODM life cycle: *Plan Domain Engineering*, *Model Domain*, and *Engineer Asset Base.*
- ***Sub-phase***: A set of tasks that collectively perform a coherent higher-level function within a phase.
- ***Task***: The low-level sets of activities where the detailed ODM work is performed and the workproducts are produced.

The ODM life cycle is described in detail in Section 4. The *Plan Domain Engineering*, *Model Domain*, and *Engineer Asset Base* phases are described in Sections 5, 6, and 7, respectively. The sub-phases and tasks within each of the phases are each described in individual subsections organized hierarchically within Sections 5, 6, and 7.

The phase, sub-phase, and task sections each include a process tree diagram for the current phase, with shaded areas showing which portions of the phase are described by the section. These diagrams not only show the scope of each section, but also serve as recurring roadmaps to help readers determine their "location" within the process model. For example, Exhibit 6 below shows the diagram from Section 5.1, which describes the *Set Project Objectives* sub-phase.

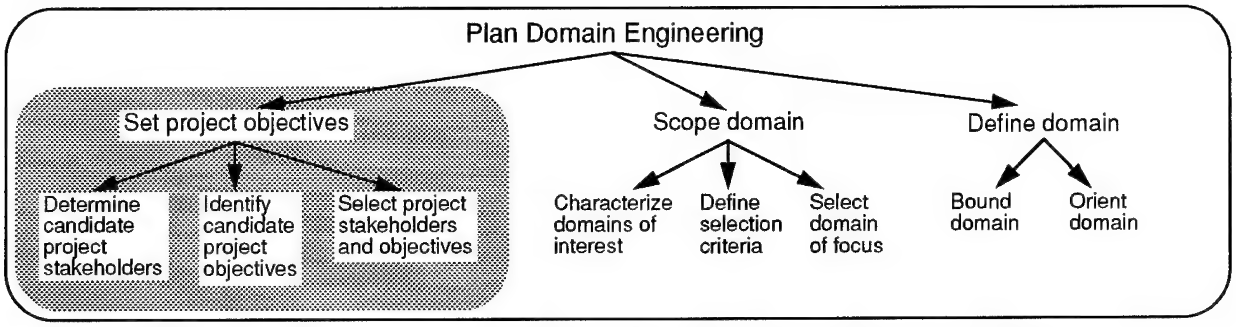

Plan Domain Engineering
Set project objectives
Scope domain
Define domain
Determine candidate project stakeholders
Identify candidate project objectives
Select project stakeholders and objectives
Characterize domains of interest
Define selection criteria
Select domain of focus
Bound domain
Orient domain

**Exhibit 6.** Example Process Tree Embedded within a Process Description Section

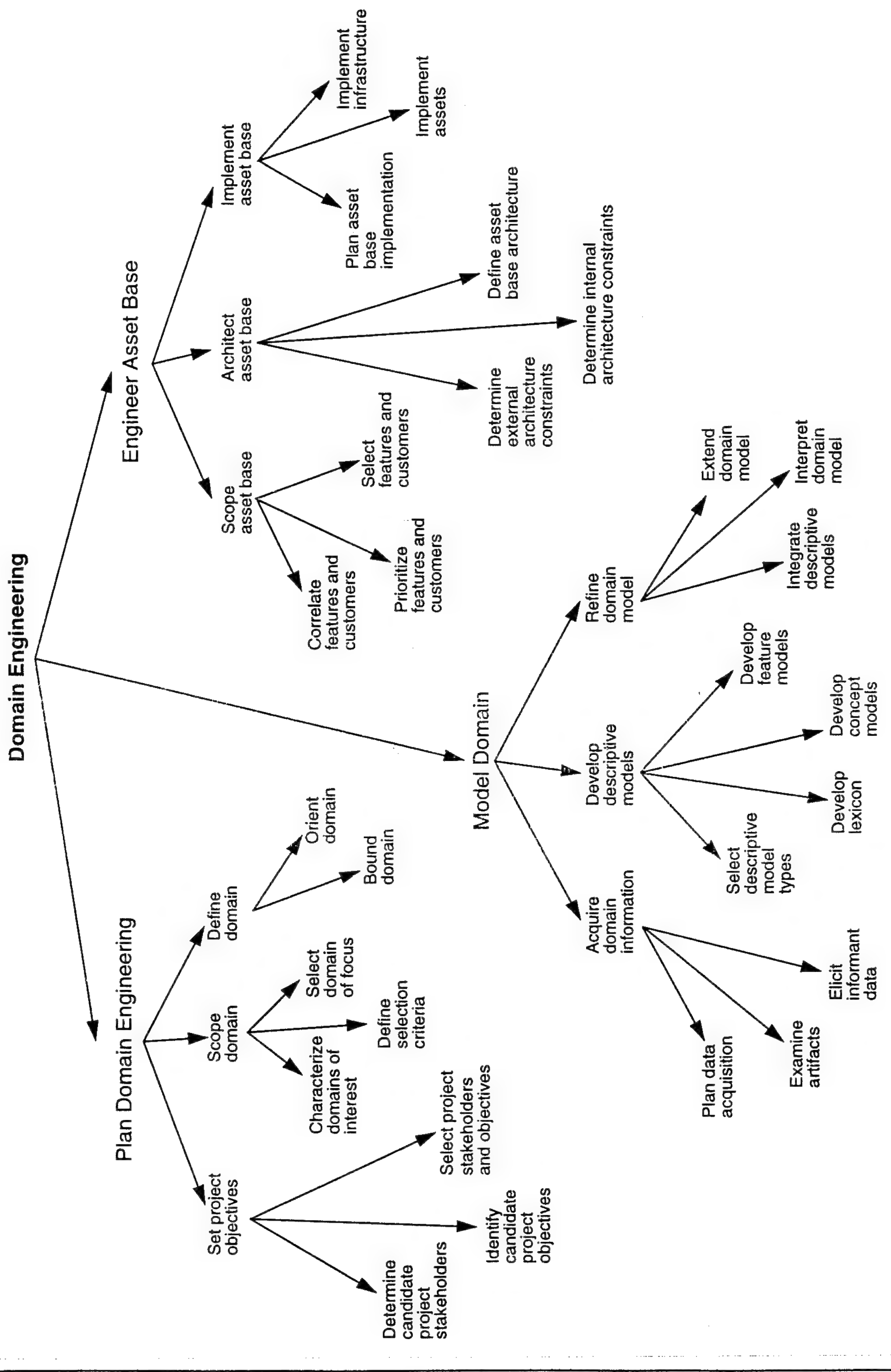

Domain Engineering
Plan Domain Engineering
Set project objectives
Determine candidate project stakeholders
Identify candidate project objectives
Select project stakeholders and objectives
Scope domain
Characterize domains of interest
Define selection criteria
Select domain of focus
Define domain
Bound domain
Orient domain
Model Domain
Acquire domain information
Plan data acquisition
Examine artifacts
Elicit informant data
Develop descriptive models
Select descriptive model types
Develop lexicon
Develop concept models
Develop feature models
Refine domain model
Integrate descriptive models
Interpret domain model
Extend domain model
Engineer Asset Base
Scope asset base
Correlate features and customers
Prioritize features and customers
Select features and customers
Architect asset base
Determine external architecture constraints
Determine internal architecture constraints
Define asset base architecture
Implement asset base
Plan asset base implementation
Implement assets
Implement infrastructure

**Exhibit 5.** Domain Engineering Process Tree

The process trees present a simplified view of the more detailed structure of the process model, which is represented in IDEF$_0$ diagrams. The life cycle, phase, and sub-phase sections each include an IDEF$_0$ diagram showing the information flows among the processes at the next lower level. For example, each sub-phase section includes an IDEF$_0$ diagram showing the information flows among the tasks in that sub-phase. Appendix A includes the entire ODM IDEF$_0$ process model. If you are unfamiliar with IDEF$_0$, please consult the appendix for a brief introduction to the IDEF$_0$ notation.

The general approach taken within the process description sections is to present information at the lowest section level at which it applies, while minimizing redundancy across sections. For example, low-level details about ODM activities and workproducts are presented in the task sections, whereas information about how the tasks within a given sub-phase interrelate (e.g., sequencing considerations) is presented in the sub-phase section. The higher level sections also discuss the more general and strategic considerations involved in applying ODM.

## Section Structure

The life cycle, phase, and sub-phase sections include the following information:

- **Purpose**: The overall objectives and benefits of the process.
- **Description**: An overview of what the process does and, to whatever extent is necessary, how it does it (emphasizing data flows and interactions among the lower level processes, rather than activities that occur within those processes, since they are described in detail in subsequent sections). The description may also introduce concepts that are needed to understand the lower level processes.
- **Sequencing**: Guidelines regarding the sequence in which lower level processes can be performed. In general, ODM processes needn't be performed in a particular sequence. Issues such as iteration and parallelism are addressed here.

The Description and Sequencing discussions typically elaborate on the process tree and IDEF$_0$ diagrams included with the section or use them to illustrate points about the process.

The task sections (e.g., 5.1.1, *Determine Candidate Project Stakeholders*) constitute the bulk of the process model description. These sections describe the low-level ODM activities, workproducts, and information flows in detail and also offer tactical guidance for regulating and controlling the processes. The task sections are organized as follows:

- **Purpose**: The objectives and benefits of the task.
- **Entrance Criteria**: A set of conditions that should be satisfied before the task can begin.
- **Inputs**: Information that the task accesses and manipulates in performing its function.
- **Controls**: Information that regulates or controls how the task is performed.
- **Activities**: The specific actions or steps to perform in carrying out the task and producing the workproducts. In general, the activities need not be performed strictly in the order they are listed, although this can vary significantly from task to task.
- **Workproducts**: The key results of the task.
- **Guidelines**: Hints, suggestions, and criteria for performing the task effectively.

- **Exit Criteria**: A set of conditions for determining when the task is completed.
- **Validation and Verification**: Criteria and techniques that can be applied to determine the completeness, consistency, or quality of the workproducts.

The Inputs, Controls, and Workproducts directly reflect the input, control, and output data items on the $IDEF_0$ diagrams. Most of the data items in the diagrams are the workproducts of ODM processes, although some inputs to the overall process originate outside of ODM.

The full set of Workproduct descriptions appearing in the task sections is collected for reference purposes in Appendix C, in summary form. Templates for a selected subset of the workproducts are also included in Appendix C. These templates are intended to convey the workproducts' general structure and content. They are generally not in a form that will be directly usable as-is, but will need to be tailored to meet a project's specific needs (e.g., the number of columns and their labels will typically differ from one project to another).

## Presentation Conventions

A number of conventions were applied in writing the sections within this part to make the process model descriptions easier to read and understand. These conventions include:

- **Section subheadings**:

  All of the first-level section subheadings (Purpose, etc.) are in bold on a line by themselves. For example:

  **Purpose**

  These subheadings are also in a slightly larger font than regular text paragraphs.

- **Information under section subheadings**:

  — Purpose, Description, and Sequencing:

  The information under these subheadings is prose paragraphs, in whatever format is most appropriate for the material.

  — Activities:

  Each activity associated with the process is signified by a subheading on a line by itself in the following format:

  ➤ <u>Analyze Usability</u>

  Under each of these subheadings, there may be subordinate subheadings representing subactivities. These subheadings look like:

  *Product Line Segmentation*

  — Workproducts:

  Each workproduct produced by the process is signified by a subheading on a line by itself in the following format:

  ■ <u>DOMAIN MODEL</u>

Substructure within the workproducts is generally shown using bulleted and sub-bulleted items underneath the subheadings.

— Inputs and Controls:

These are presented as bulleted lists. Each item in the list includes the $IDEF_0$ data item name in an underlined run-in heading, followed by some explanation of how the item relates to the process. E.g.:

- DOMAIN MODEL. This process uses the DOMAIN MODEL to . . .

— Guidelines and Validation and Verification:

These are also presented as bulleted lists. Each item typically includes an underlined phrase (usually a run-in heading, but not always) that concisely summarizes the item, followed by explanatory text. E.g.:

- Document model boundary issues/decisions. Negotiating clear boundaries . . .

Under these subheadings, there may also be non-bulleted prose paragraphs providing sweeping or summary criteria.

— Entrance and Exit Criteria:

The format for these is less consistent than for the other subheadings. They are sometimes expressed as bulleted lists (like Guidelines) and sometimes as prose paragraphs.

- **Typographical conventions**:

— SMALL CAPS: ODM workproducts (initial large caps) and other $IDEF_0$ data items (all small caps)

— *Italic With Initial Caps*: ODM process names

— ***bold italic***: An instance of a key term in the ODM lexicon (usually this is reserved for the initial instance of the term within some cohesive portion of the guidebook, such as a section or group of sections)

— *italic*: Emphasizes or highlights any term in running text.

— **bold**: Used only in section headings and subheadings.

# 4.0 Domain Engineering Life Cycle

This section introduces the basic ODM process scenario and some of the key terminology used throughout the remaining sections in Part II. The basic sequence described in this document is a single-project scenario, producing a domain definition, domain model, and domain assets for a single domain of focus. Note that in some projects, it may not be necessary to complete this full scenario. Some possible ways of tailoring this basic sequence to meet specific project needs are discussed in Section 10.

## The Domain Engineering Project Context

***Domain engineering*** involves the creation of new technological and human systems for managing domains within some ORGANIZATION CONTEXT, potentially transforming both the organizations and technical systems within that context. The process tree depicted in Exhibit 5 (in the introduction to Part II above) shows the scope of a domain engineering project in terms of the processes involved. In the following descriptions, this overall project scope is called the ***domain engineering life cycle***. This life cycle as presented here reflects the ODM approach to domain engineering, but is valid to some degree for domain engineering in general.

Domain engineering takes place in the broader context of a *reuse program*, which is described more fully in the STARS Conceptual Framework for Reuse Processes (CFRP) model [CFRP93a]. The CFRP outlines a taxonomy of reuse processes intended to span all reuse-specific aspects of reuse-based software development. The CFRP model (shown at a high level in Exhibit 7) is partitioned into two major submodels, known as *idioms*: Reuse Management (planning, enactment and learning) and Reuse Engineering (creating, managing and utilizing reusable software assets).

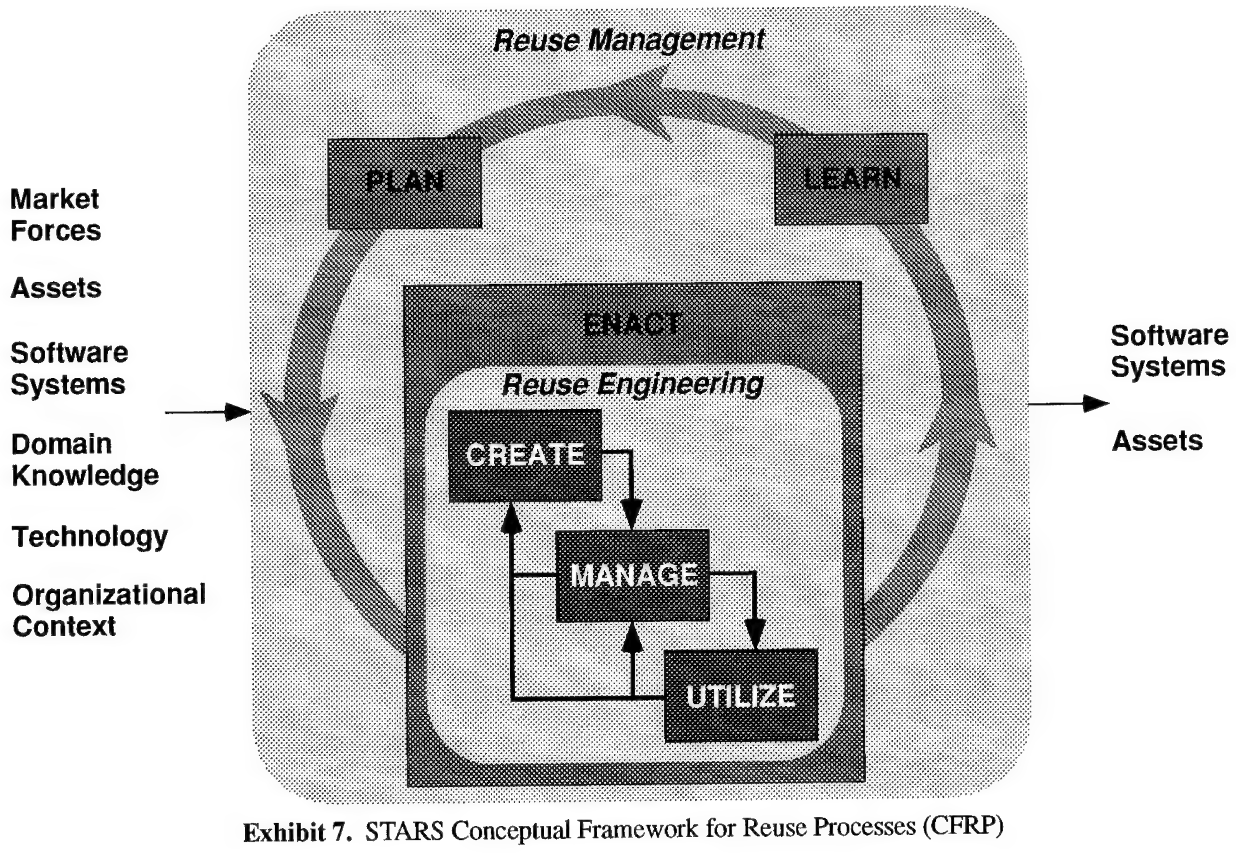


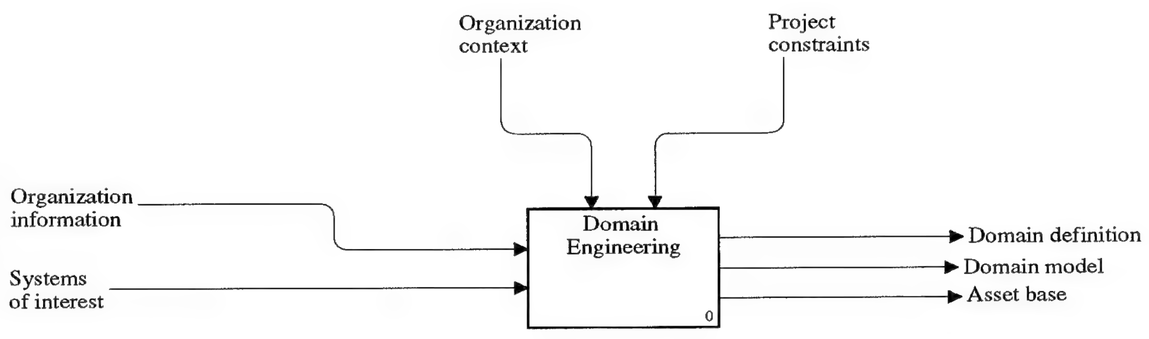
**Exhibit 7.** STARS Conceptual Framework for Reuse Processes (CFRP)

In CFRP terms, ODM focuses primarily on the Asset Creation (i.e., domain engineering) process family of the Reuse Engineering idiom. ODM's *Plan Domain Engineering* phase can be considered a specialization of the processes called out in the CFRP's Reuse Planning family of processes (with the exception of Infrastructure Planning and Project Planning, which are dealt with, respectively, in the Supporting Methods described in Section 8 and the Guidelines for Applying ODM in Section 10). CFRP Learning processes are encapsulated in a separable Learning layer of ODM, described in Section 9.

Domain engineering does not include the ongoing management of the domain model and asset base, nor utilization of the asset base by application development projects (i.e., the CFRP Asset Management and Asset Utilization processes, respectively). Relationships between the domain engineering project and other efforts, such as system reengineering projects or planned new products, must be carefully considered in planning and managing the overall reuse program. Discussing these relationships in detail is beyond the scope of this document.

Exhibit 8 shows the IDEF$_0$ context diagram for an ODM domain engineering project initiated within some ORGANIZATION CONTEXT. Depending on the nature of the domain engineering initiative and funding for the project, the project context can be:

- a single organization or a division of a larger organization, such as a corporate or university research and development facility, a product-line division, or a system development group,
- multiple organizations, as typified by standards organizations and consortia with common interests in particular functional areas,
- a marketplace of varied competitor, partner and customer organizations, or



Viewpoint:

Audience:

Stakeholders:

Purpose:

**Exhibit 8.** Domain Engineering IDEF$_0$ Context Diagram

- a technical community.

This set of stakeholder interests and relations forms the ***stakeholder context*** for the domain engineering project. Descriptions may informally refer to this context as "the organization" for convenience.

The two primary inputs to *Domain Engineering* are ORGANIZATION INFORMATION and SYSTEMS OF INTEREST. ORGANIZATION INFORMATION includes:

- human domain expertise,
- textbooks, and
- industry sources of information.

The project produces three key results: a DOMAIN DEFINITION, a DOMAIN MODEL, and an ASSET BASE. An ***asset*** may be a software component, a generative tool, a template for a design document, or any workproduct of the software life cycle specifically engineered for reuse within a domain-specific scope of applicability. The ***asset base*** is the full set of ASSETS for a domain, together with the ASSET BASE ARCHITECTURE that integrates the assets and ASSET BASE INFRASTRUCTURE required for the domain.

## Phases of Domain Engineering

Domain engineering consists of three main ***phases***, as shown in Exhibit 9: *Plan Domain Engineering*, *Model Domain,* and *Engineer Asset Base.* These phases are called simply (Domain) Planning, (Domain) Modeling, and (Asset Base) Engineering in the following paragraphs for convenience.

The primary purpose of the Planning phase is to set objectives and select and scope a domain for the project in alignment with overall organizational needs. Explicitly modeling the various stakeholders for the project in the DOMAIN STAKEHOLDER MODEL is a key task to ensure this alignment. PROJECT OBJECTIVES and a DOMAIN DEFINITION are the other key outputs of this phase. The Planning phase includes considerable definition work beyond simple selection of the domain, to ensure that the domain is appropriate for detailed modeling. Obtaining commitment of the various stakeholders involved is also a key task of this phase.

The primary purpose of the Modeling phase of the ODM life cycle is to produce a DOMAIN MODEL for the selected domain, based on the DOMAIN DEFINITION produced in the Planning phase. The key role of the DOMAIN MODEL is to describe common and variant features of systems within the domain, with rationale for the variations. The DOMAIN MODEL is subsequently used in the Engineering phase as a basis for selecting the range of variability to be supported by assets in the ASSET BASE. A secondary product of this phase is the DOMAIN DOSSIER which documents the specific information sources used as a basis for modeling. The DOMAIN DOSSIER may also be used during the Engineering phase to trace legacy artifacts that are candidates for reengineering into reusable assets; or to identify constraints in systems into which assets may be migrated.

The primary purpose of the Engineering phase of ODM is to scope, architect, and implement an ASSET BASE that supports a subset of the total range of variability encompassed by the DOMAIN MODEL, a subset that addresses the domain-specific requirements of a specific set of customers. The key benefit of the ASSET BASE produced in this phase is that it achieves an overall economy in the cost of developing systems in the domain, when viewed from the perspective of the customer contexts for the ASSET BASE taken as a whole. The key challenge of this phase is identify-

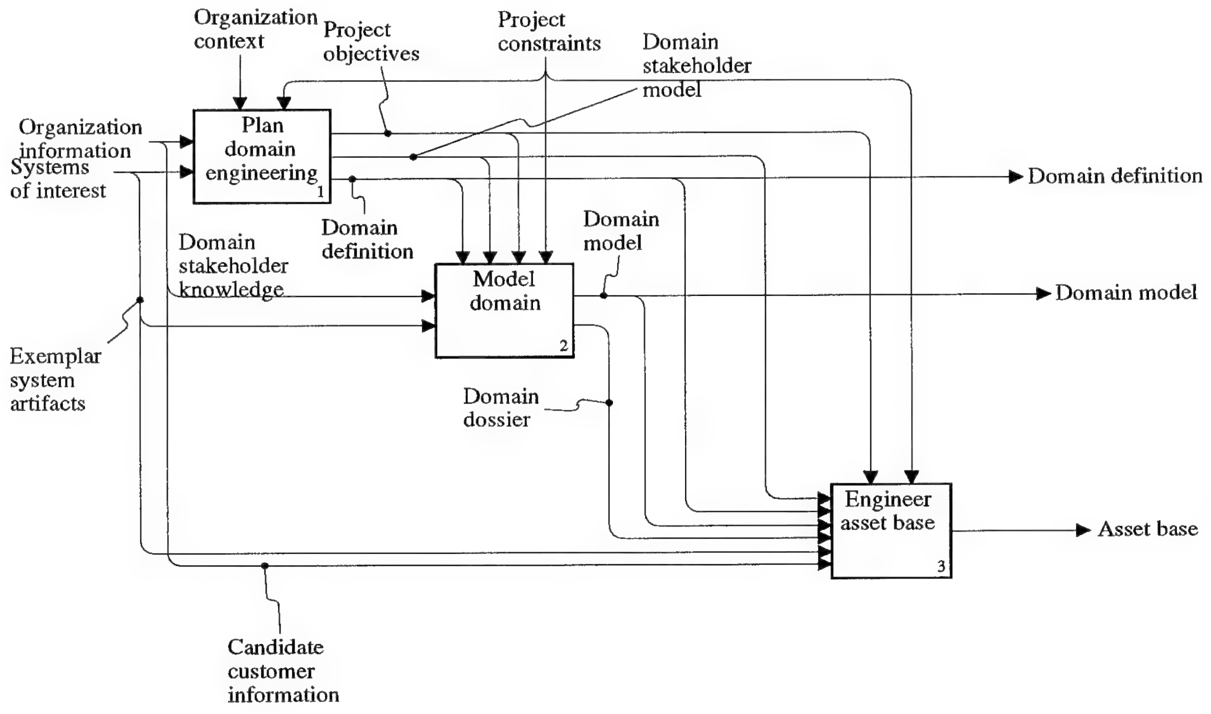


**Exhibit 9.** Domain Engineering IDEF$_0$ Diagram

ing an appropriate market of customers and the features required to support this market, to make the ASSET BASE a viable and ongoing structure for achieving increasing levels of reuse. An additional challenge in the Engineering phase is to manage the potentially explosive variability in the domain to eventually produce tractable specifications for implementors to create the asset base infrastructure and individual assets.

## Sequencing

The ODM process structure allows for iteration and re-entry to accommodate both unforeseen risks and follow-on opportunities. This will most often occur as later extensions to the domain engineering project or subsequent projects within the same organization. Subsequent projects initiated in the same organizational context can select new domains of focus, reusing some workproducts from previous projects. By choosing different data as a basis for modeling, different domain models can be produced starting from a common domain definition. Similarly, a single domain model can provide the starting point for multiple asset base engineering efforts. Within each phase various workproducts and process checkpoints allow for efficient iteration if required.

It is also possible to manage projects with varying degrees of parallelism involved. This may allow for better use of resources, but will significantly increase both the technical and management complexity of the projects. Alternative strategies for project management of the overall domain engineering life cycle can be considered, including parallel, pipelined, or iterative strategies. In any sequence, however, a core principle of ODM is to maintain clarity about whether descriptive or prescriptive modeling is being performed, and to keep the resulting models separate.

# 5.0 Plan Domain Engineering

## Purpose

A domain engineering project involves negotiating a complex set of relations among customer application groups. The primary goals of *Plan Domain Engineering* are to ensure that both overall project objectives and the selected domain of focus are appropriately scoped and aligned with the broader strategic needs of stakeholder organizations. This depends on establishing an explicit model of the ORGANIZATION CONTEXT for the project and the domain. A key challenge in this phase is the fact that this context will often involve multiple organizations or organization divisions, with both coinciding and conflicting interests.

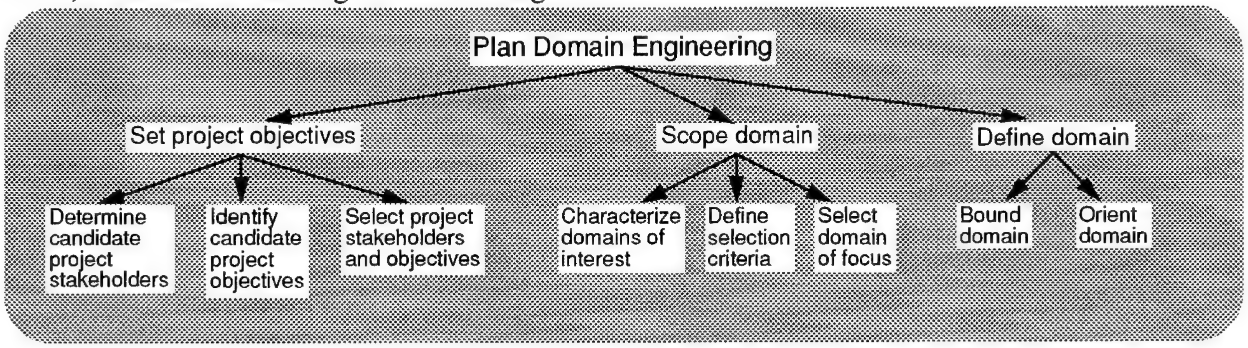

Plan Domain Engineering
Set project objectives
Scope domain
Define domain
Determine candidate project stakeholders
Identify candidate project objectives
Select project stakeholders and objectives
Characterize domains of interest
Define selection criteria
Select domain of focus
Bound domain
Orient domain

**Exhibit 10.** Plan Domain Engineering Process Tree

## Description

As shown in Exhibit 11, there are three sub-phases in *Plan Domain Engineering*. An overall control to the phase is the ORGANIZATION CONTEXT in which the project is initiated. Part of this context is an explicit or implicit PROJECT CHARTER that establishes an overall mandate for the project; it may be formal or informal.

- The first sub-phase, *Set Project Objectives,* defines objectives in relation to candidate and selected project stakeholders, documented in an explicit set of PROJECT OBJECTIVES and a PROJECT STAKEHOLDER MODEL. The process does not assume that the PROJECT CHARTER constitutes final commitment to perform the project; obtaining key stakeholder commitment to the objectives is a critical element of this sub-phase.
- The primary output of the second sub-phase, *Scope Domain,* is the DOMAIN SELECTION. Also, in this sub-phase planners refine the PROJECT STAKEHOLDER MODEL into the DOMAIN STAKEHOLDER MODEL. Information about other domains relevant within the project context is captured in the DOMAINS OF INTEREST report.
- The primary purpose of the third sub-phase, *Define Domain,* is to formalize and validate the loose definition of the domain resulting from the *Scope Domain* sub-phase. This process, which can almost be thought of as performing a rapid prototype of the entire *Model Domain* phase to follow, involves several tasks and workproducts uniquely emphasized in ODM. Final decision to proceed to detailed domain modeling is not made until this definition is complete.

At the conclusion of one iteration of Domain Planning, we have selected and defined a ***domain of focus*** for the domain engineering project. The major decision point at the conclusion of these steps is whether to continue on to the *Model Domain* phase with the domain of focus as currently

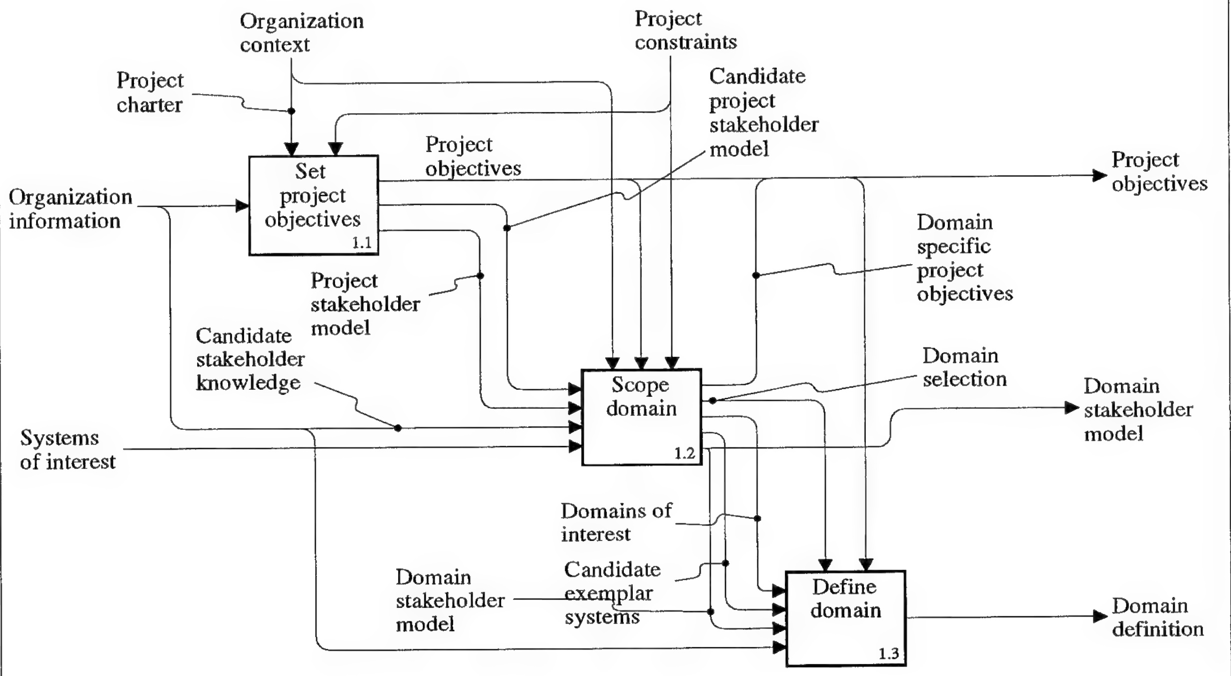


**Exhibit 11.** Plan Domain Engineering $IDEF_0$ Diagram

defined, or to recursively perform *Plan Domain Engineering* to more narrowly focus the selected domain. (See the discussion of Sequencing below.)

The name of this first phase of the ODM life cycle may suggest that it serves a conventional project planning function. Though these activities do need to be carried out in concert with the activities in this phase, the core ODM process model excludes most typical project planning and infrastructure planning activities. The process model and the sections describing it that follow focus primarily on differentiating aspects of domain engineering project planning. Guidelines on how to appropriately tailor the ODM life cycle and select supporting methods, includes selecting appropriate domain modeling methods and tools, project staffing, training, and establishing organization structures for the project, are discussed briefly in Section 10.0.

## Sequencing

- Selecting objectives and domain. The task structure as defined is based on experience that it can be a useful exercise to document explicit project objectives *before* selecting the domain of focus. In practice, many project teams will skip over performing *Set Project Objectives* in a formal way and rush to the *Select Domain of Focus* task (within the *Scope Domain* sub-phase in the ODM process model). They then discover that many implicit objectives surface when domain choices raised them indirectly as issues.

  Even after a good-faith attempt is made to document PROJECT OBJECTIVES first, some iteration will be inevitable. But the attempt will still be of value in clarifying hidden agendas and constraints on the project.

- Expect iteration in domain scoping and definition. Domain scoping and definition may reveal stakeholders not anticipated in the initial project context, e.g., potential affiliations with other parts of the organization or other organizations. These changes might necessitate revisiting the original steps.

  As with any iterative task, stay alert for process breakdowns like looping incessantly without coming to closure. Emphasize updating initial workproducts where appropriate and then move on.

Domain of focus is mandated.

In certain situations the domain of focus will appear to be a given in the PROJECT CHARTER. This might suggest that tasks up to and including domain selection are superfluous and can be skipped, and that domain engineering can begin with the *Define Domain* task.

There are reasons to be cautious in this approach. Since domain engineering is still an immature discipline there are likely to be significant misunderstandings (or, more properly, divergent understandings) of what constitutes a domain, what makes a domain a good choice for reuse, etc. There is still a need, therefore, to validate the domain selection against organization needs, and to identify possible synergies and/or conflicts with different stakeholders, and clarify an explicit set of project objectives. The need to do this validation may become evident later on, where conflicts or breakdowns in getting consensus will provide insight about missing or undocumented criteria.

The sequence of the processes can certainly vary, however. Planning processes prior to domain selection can be performed as a review or validation of the selected domain, perhaps even in parallel with *Define Domain* activities. There may also be less value in separating certain steps and workproducts (e.g., DOMAINS OF INTEREST can be identified primarily in the course of creating the DOMAIN INTERCONNECTION MODEL).

In addition, the mandated domain may be too large to address given project constraints. In this case, *Scope Domain* and *Define Domain* can be performed iteratively to progressively narrow the domain scope. However, this cannot be done effectively if the initial context-setting steps are skipped.

## 5.1 Set Project Objectives

### Purpose

There can be many motivating factors behind an organization's decision to explore domain engineering. Specific objectives depend to a large extent on the role of software technology within stakeholder organizations. Often the business context and rationale for a domain engineering project are not explicitly documented at the start of the project. The goal of the *Set Project Objectives* sub-phase is to explicitly document PROJECT OBJECTIVES and characterize them with respect to the interests of an explicit set of project stakeholders.

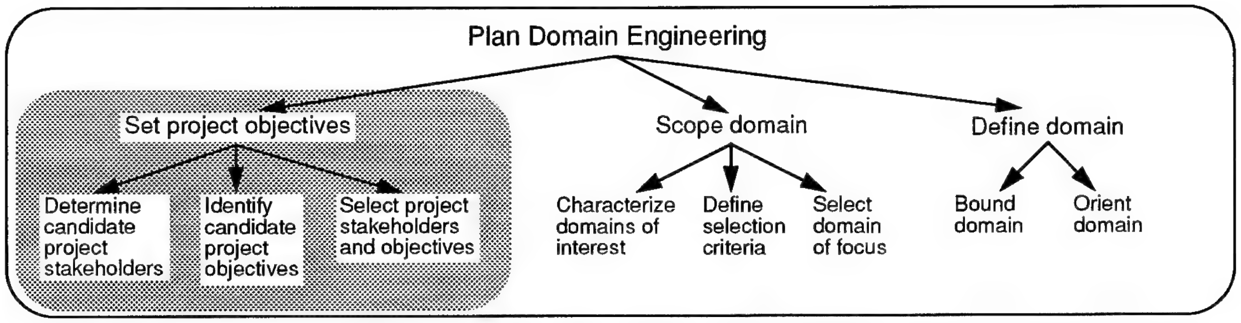

Plan Domain Engineering
Set project objectives
Scope domain
Define domain
Determine candidate project stakeholders
Identify candidate project objectives
Select project stakeholders and objectives
Characterize domains of interest
Define selection criteria
Select domain of focus
Bound domain
Orient domain

**Exhibit 12.** Set Project Objectives Process Tree

### Description

As shown in Exhibit 13, the *Set Project Objectives* sub-phase consists of three tasks, *Determine Candidate Project Stakeholders*, *Identify Candidate Project Objectives*, and *Select Project Stakeholders and Objectives.*

- The first task, *Determine Candidate Project Stakeholders*, results in a model of candidate stakeholder organizations and their interests relative to the project.
- In the second task, *Identify Candidate Project Objectives*, planners determine candidate objectives and map them against identified stakeholder interests.
- In the *Select Project Stakeholders and Objectives* task a set of objectives are converged upon that are internally consistent, satisfy key stakeholder interests, and are feasible given PROJECT CONSTRAINTS.

### Sequencing

- The *Determine Candidate Project Stakeholders* and *Identify Candidate Project Objectives* tasks can be performed sequentially or in parallel, iterating between them as additional stakeholders and objectives are identified based on current stakeholders and objectives.
- Since the *Select Project Stakeholders and Objectives* task involves trade-off analysis between stakeholder interests and objectives, both the *Determine Candidate Project Stakeholders* and *Identify Candidate Project Objectives* tasks must be completed before the *Select Project Stakeholders and Objectives* task begins.

These sequencing constraints reflect a element in this task structure which will be seen to recur throughout the ODM process model. The sub-phase is structured in terms of two initial *descriptive* tasks that emphasize generation of exploratory and divergent views, followed by a *prescrip-*

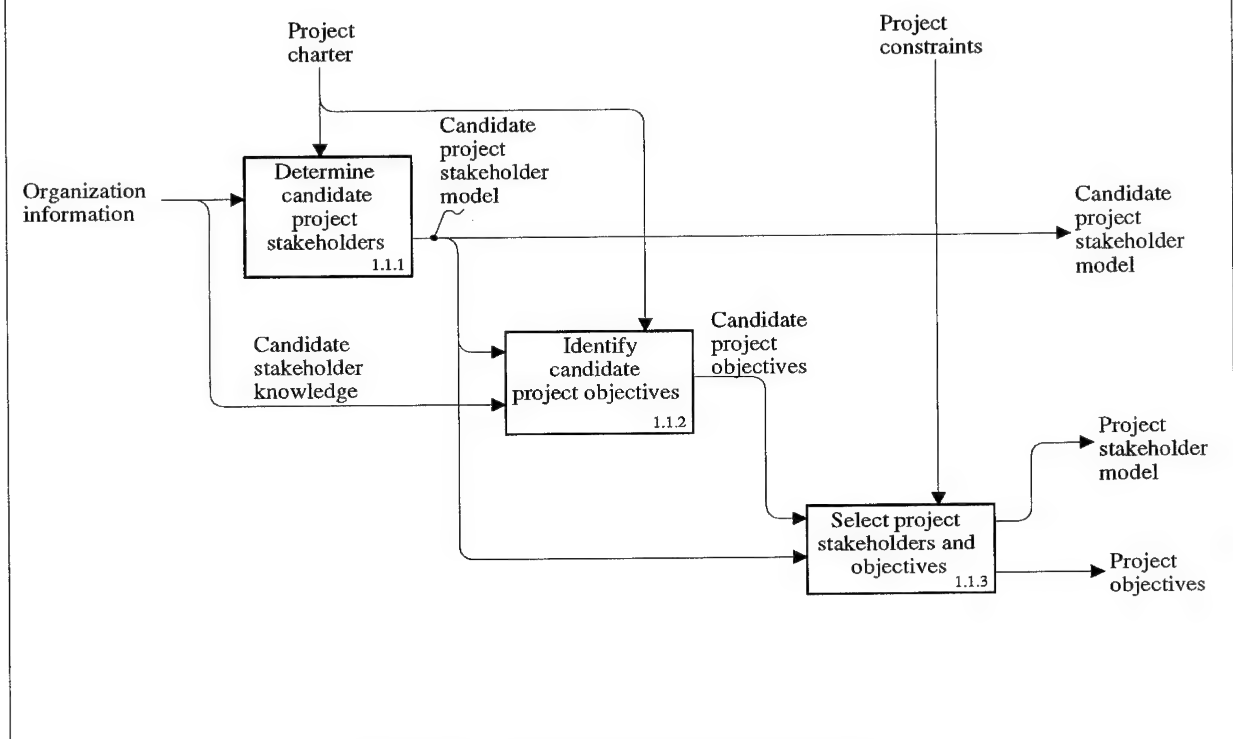

Project charter
Project constraints
Organization information
Determine candidate project stakeholders 1.1.1
Candidate project stakeholder model
Candidate project stakeholder model
Candidate stakeholder knowledge
Identify candidate project objectives 1.1.2
Candidate project objectives
Project stakeholder model
Select project stakeholders and objectives 1.1.3
Project objectives

**Exhibit 13.** Set Project Objectives IDEF$_0$ Diagram

*tive* task emphasizing convergence on commitments and decisions (*Select Project Stakeholders and Objectives*). The sub-phase thus enacts an initial spiral of the *descriptive-prescriptive* movement reflected in the overall ODM life cycle. A similar rhythm will be described in the following sub-phase, *Scope Domain.*

## 5.1.1 Determine Candidate Project Stakeholders

### Purpose

The primary goal of this task is to analyze characteristics, relationships between, and interests of candidate project stakeholders. ***Candidate project stakeholders*** are parties whose commitment and buy-in is critical to the success of the domain engineering project. The main function is to derive a model of these stakeholders, called the CANDIDATE PROJECT STAKEHOLDER MODEL, from overall information about the project's ORGANIZATION CONTEXT. This information may be well-documented in an explicit PROJECT CHARTER, or may be informal and implicit, requiring a fair amount of analysis and reflection to articulate.

Stakeholder analysis is a useful starting point for any system engineering effort. In domain engineering, it is particularly essential (and complex) because stakeholders are likely to span project, functional, departmental and organization boundaries. The resulting stakeholder model provides an initial basis for setting concrete and achievable objectives for the reuse program as a whole; these in turn provide a basis for selecting and defining an appropriate domain of focus. The stakeholder model and associated assessment information is used and further refined throughout the

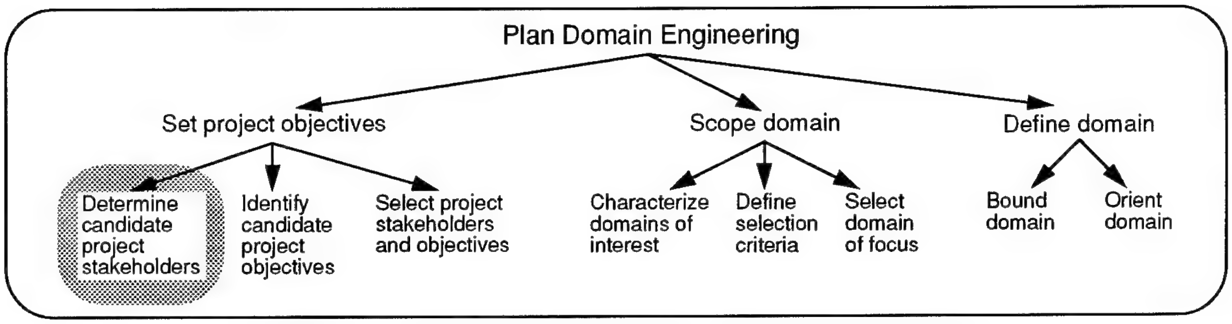

Plan Domain Engineering
Set project objectives
Scope domain
Define domain
Determine candidate project stakeholders
Identify candidate project objectives
Select project stakeholders and objectives
Characterize domains of interest
Define selection criteria
Select domain of focus
Bound domain
Orient domain

**Exhibit 14.** Determine Candidate Project Stakeholders Process Tree

domain engineering life cycle, and can also provide a useful starting point for subsequent domain engineering projects in the same ORGANIZATION CONTEXT.

## Entrance Criteria

- An initiative has been launched for a domain engineering project.
- The mandate for the project is documented in the PROJECT CHARTER; or the planning team has documented the assumed or implied charter and verified it with the project initiators.
- No domain has been specified as a constraint of the project. There *will* often be an overall business area specified that indicates the organization entities considered to be within the scope of the project. The risk is greater if a specific domain has been mandated, since characteristics of the mandated domain may prove incompatible with other explicit or implicit PROJECT CONSTRAINTS (such as the use of a particular technology).

## Inputs

- ORGANIZATION INFORMATION. Information about stakeholder organizations, used to develop the model of stakeholder roles, interests, etc. General sources of ORGANIZATION INFORMATION may include: organization charts, business plans, marketing literature, annual reports, and informal knowledge of project members. Other valuable sources of information are results of reuse-specific and broader organization assessments.

  Informal knowledge of project members should be used cautiously, particularly when the project team has a different background and culture from prospective customer groups; e.g., when a research and development-oriented group will be doing modeling to create assets for an application engineering group.

## Controls

- PROJECT CHARTER. Provides bounding criteria for the ORGANIZATION CONTEXT of the project, including whether the initiative is part of a more comprehensive reuse program. The charter may be a formal document or may be implicit, based on the informal contextual knowledge of project members.

Note that PROJECT CONSTRAINTS are *not* a control to this task. Don't screen stakeholder interests based on project resources. Here the intent is to identify as many reinforcing and competing interests as possible. Screening of stakeholder interests is done in the *Select Project Stakeholders and Objectives* task.

## Activities

General organization assessments may prove useful to subsequent projects, especially if the CANDIDATE PROJECT STAKEHOLDER MODEL reflects a stable organization boundary. However, assessing and modeling the entire organization is too broad a task for the purpose of setting PROJECT OBJECTIVES. Utilize general assessment information where available, but keep the project focus in mind. These related techniques are discussed in Stakeholder Analysis in Section 8.1

➤ Identify candidate project stakeholders and their roles

Utilizing ORGANIZATION INFORMATION and the PROJECT CHARTER determine relevant candidate project stakeholders and their roles and relationships. ORGANIZATION INFORMATION that is used includes information from known stakeholders and project members' informal knowledge. Candidate project stakeholders are the most diverse group of people that can be considered to have an interest in the software systems and domains contained within the ORGANIZATION CONTEXT. They may include managers, software engineers, test developers, field technicians, technical support staff, documentation and training staff, end users, value-added resellers, marketers, and competing organizations. If technology selections have been made, technology developers should also be considered as stakeholders. Stakeholders who founded the PROJECT CHARTER (presumably the funders) are usually key stakeholders. There will be a unique configuration of stakeholders for each ORGANIZATION CONTEXT in which domain engineering is initiated. Stakeholders can play multiple roles. Exhibit 15 shows a generic set of stakeholder roles and interrelationships.

Both of the methods below should be used to identify candidate stakeholders and their roles to ensure that no candidate stakeholders are overlooked:

- Brainstorm candidate stakeholders and then determine the roles held by these stakeholders.
- Use the key questions below to determine relevant roles and then identify the stakeholders who fill these roles.
  - Who are core or key customers of the domain engineering project?
  - Who is funding the project?
  - Who defines success for the project? (e.g., the funder, the funder's customers?)
  - Who will be performing the project?
  - Who are potential informants for Domain Modeling?
  - Who are potential customers for the asset base developed during Asset Base Engineering?
  - Based on organization relations (e.g., external market relations, internal customer-supplier relationships), which stakeholders are potential asset developers, asset base managers, utilizers of assets?

As an example, candidate project stakeholders for a manufacturer of software-intensive peripheral equipment such as laser printers might include:

- Software developers,
- Integration testers,

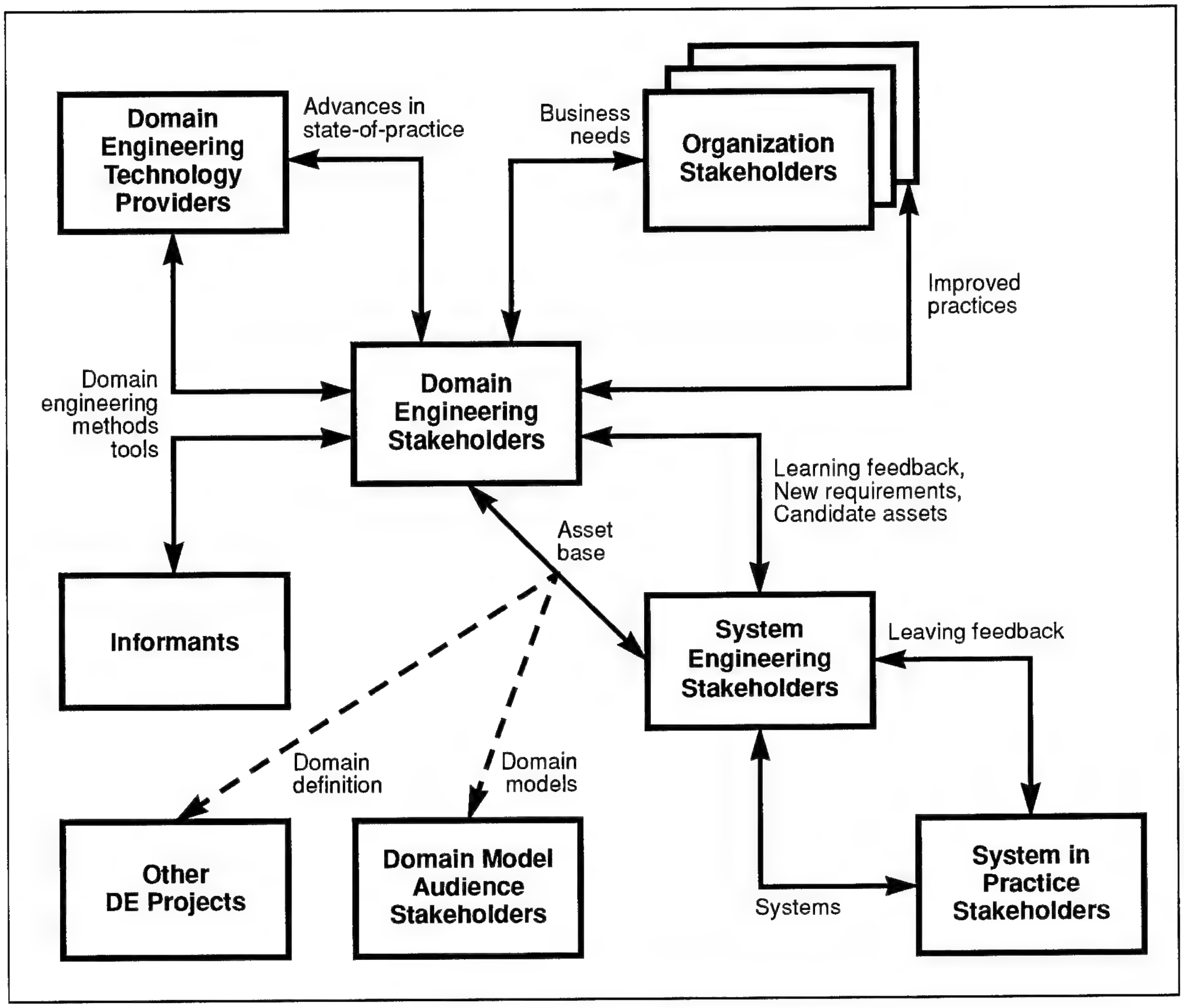

Domain Engineering Technology Providers
Advances in state-of-practice
Business needs
Organization Stakeholders
Improved practices
Domain engineering methods tools
Domain Engineering Stakeholders
Learning feedback, New requirements, Candidate assets
Asset base
Informants
System Engineering Stakeholders
Leaving feedback
Domain definition
Domain models
Other DE Projects
Domain Model Audience Stakeholders
Systems
System in Practice Stakeholders

**Exhibit 15.** Domain Engineering Stakeholder Roles

- Field technicians from the producer organization,
- Customer system managers who might install new software upgrades,
- Third-party software vendors who might write driver software to make use of the peripheral's software, and
- End users of the equipment.

Once both stakeholders and stakeholder roles have been identified, cross-check them using a workproduct like the PROJECT STAKEHOLDER ROLES template illustrated in Exhibit C-1. This template allows the identification of stakeholders that play multiple roles, and roles that are filled by multiple stakeholders. Validate that all critical roles are accounted for, and that all key stakeholders have a defined role.

➤ Characterize stakeholder organizations

Characterize stakeholder organizations, at a minimum, in terms of these criteria:

- Sector orientation:
  - — government organization,
  - — government contractor,
  - — commercial software product developer,
  - — service organization, etc.

  Sector orientation strongly influences stakeholder interests relevant to the domain engineering project. For example, time to market may be a significant motivator for a commercial product developer, whereas reducing maintenance costs may be the major cost driver for a government maintenance organization focused on post-deployment system support. For a research or educational institution, obtaining breakthrough research results will be of high priority. Sector orientation issues such as whether data is classified, sensitive, or public should also be considered.

- Business model:
  - — contract-based,
  - — product-based, or
  - — service-based.
- Life cycle phases supported by the organization:
  - — development,
  - — prototyping,
  - — maintenance, or
  - — IV&V.
- Role of software engineering within the organization. Does the organization produce software products, or other products with significant software components? While related to the sector typology mentioned above, these aspects are somewhat independent; for example, government organizations could perform multiple roles.
- How is software is used internally within the organization? For many large enterprises (e.g., commercial banks, airline companies) the volume of software developed to support internal business operations will justify applying a domain engineering approach. warrant a domain modeling project.

In organizations involved with software engineering products or contracts, there is traditionally a split between the engineering and internal software groups. Opportunities for domain engineering may occur in both areas. Viewing the role of software within the total organization can facilitate restructuring along the lines of knowledge domains.

➤ <u>Characterize stakeholder interests</u>

Identify stakeholder interests as they relate to the project. Interests may be expressed directly by stakeholders or inferred by the planners. Knowledge of these interests assists planners in obtaining buy-in and commitment from stakeholders at all levels. This stakeholder commitment is essential for domain data acquisition and validation, and ensures use of the assets developed. Interests assessed should include the following:

- Incentives/enablers: what factors would be perceived as motivators to the stakeholder? E.g., what are the forces driving the application groups in which reuse-based development is to be introduced?
- Risks: It is important to understand the risks to a reuse-based approach particular to each business environment in which a reuse program is initiated. Do not dismiss perceived risks too quickly as mere resistance to new technology on the part of domain engineers. There will often be sound technical reasons for skepticism that should be respected.
- Values: Individual engineers' motivation to reuse or design for reuse is often connected with core professional or aesthetic values. Domain engineering may be seen as the chance to implement the elusive "elegant general solution" that time constraints and pragmatic concerns usually prevent from being realized. This can present both opportunities and risks.

➤ Survey previous reuse efforts

Survey results of previous reuse efforts, including domain engineering projects, within stakeholder organizations. Results of previous domain engineering projects, both formal and informal, need to be considered early in the stakeholder analysis. These results can lead to the identification of key stakeholders who were involved in the previous projects. Successful efforts should have produced data useful to the new domain engineering project. War stories about unsuccessful efforts may cause stakeholders to be extremely resistant to new attempts to introduce domain engineering practices. In many cases, previous efforts will have created vested stakeholder interests, and will have a major impact on the reception of new reuse efforts.

➤ Characterize reuse practice in stakeholder organizations

Establish an accurate picture of the current state of software engineering and reuse practice within the organization. This assessment should include determining the types of reuse (informal or formal) currently being practiced and the receptiveness to future reuse efforts. The previous activity focused on those efforts identified by the organization as reuse-oriented (whether they were or not); this activity should focus on reuse practice as it is (whether recognized as such or not).

The following techniques can be used for characterizing software engineering and reuse practice:

- Utilization of relevant results from previous assessments within stakeholder organizations, when available. Assessments required for planning domain engineering may overlap to some extent with wider organization assessment initiatives including:
  - — Software engineering maturity assessments such as those performed using the SEI's Capability Maturity Model (CMM),
  - — Software productivity assessments, or
  - — Organization assessments for strategic planning, quality programs, or line-of-business initiatives.
- Assessment of the organization communication infrastructure, including internal electronic mail systems, libraries, corporate communication, training departments, and education departments. How much are these resources used for informal or ad hoc reuse? These resources will be used during domain modeling as a way to broadcast requests for domain information. Later, once an asset base is established, these same resources will be instrumen-

tal in technology transfer. Some useful questions to ask when assessing the organization communication infrastructure are:

- How are resources shared, if at all?
- What informal repositories are used?
- How do people know where to find things?
- Who are the veterans, the experts in particular subject areas that serve as informal "knowledge bases" for the developers?
- What success and/or war stories about reuse efforts are part of the culture of the development group?

- Assessment of the receptiveness of stakeholder organizations to reuse in terms of these key aspects:

  - Concepts. The exposure of stakeholders to reuse concepts and methods.
  - Capabilities. Level of maturity with regard to reuse technology and practices.
  - Beliefs. Prevalent beliefs about reuse held by the group.

  These aspects are fairly independent. An organization may perform a high level of reuse, yet be relatively unaware of formal reuse processes. Another group may be aware of reuse terminology and concepts, yet be far from practicing reuse themselves.

  A number of formal reuse assessment instruments, such as the Software Productivity Consortium's Reuse Capability Model (RCM) [VCOE92b], have been developed for this assessment function. Informal surveys may also provide an adequate foundation for this assessment step. Reuse readiness assessment is one context in which *ethnographic* data collection techniques can be used. Ethnographic data collection techniques include:

  - direct personal interviews,
  - qualitative, open-ended questions, and
  - interactions in interview settings among various individuals in the organization.

  For example, a group developing real-time applications may believe that attempts to reuse software will always result in inefficient code. This belief must be identified and understood by planners in order to arrive at objectives and measurable success criteria that will address these concerns.

## Workproducts

■ CANDIDATE PROJECT STAKEHOLDER MODEL

A list of candidate project stakeholders, partitioned into key and potential stakeholders annotated with descriptions of roles, relationships, interests, enablers and barriers, and assessments of attitudes towards reuse and reuse practices. A template for defining PROJECT STAKEHOLDER ROLES is illustrated in Exhibit C-1.

## Guidelines

- Use collaborative self-assessment techniques. There are a number of assessment instruments available that utilize an external auditing style of information gathering. Results of such assessments can be useful inputs to the *Determine Candidate Project Stakeholders* task. If assessment activities are undertaken as part of the domain engineering project, it is best to use a collaborative self-assessment style of information gathering. Outside facilitation of the assessment activity can help by providing the element of insider-outsider dialogue that will prove useful throughout Domain Modeling.

- Model diversity as well as consensus. If there are multiple project customers, document interests unique to a customer as well as common interests. Also document apparent conflicts among stakeholder interests.

- Consider competitive interests. Stakeholders may have competitive relationships, especially in multi-organization project contexts. These competitive relationships must be identified in order to develop achievable project objectives. For example, for an industry consortium, an open system standard may be a feasible outcome of domain engineering, whereas a shared component base may be infeasible because of competitor company interests.

- Consider interests that may change the business model. Some possible domain engineering objectives could change, expand or diversify the business model of stakeholder organizations. In particular, domain models or information extracted from them can become products in their own right. For example, domain analysis results could be marketed as a comparative survey, a strategic marketing plan, a competitive analysis within the domain, a training instrument, a tool to aid in experts' collaboration and sharing of domain knowledge, a database or a reference guide. For companies unaccustomed to this perspective, the shift from being product or service providers to information and knowledge providers may have profound repercussions.

## Exit Criteria

- All stakeholders whose interests could determine success criteria for the project have been included in the CANDIDATE PROJECT STAKEHOLDER MODEL.

- Additional stakeholders have been identified and have led to the discovery of opportunities for leverage and/or synergy and possible risk factors.

- Potential conflict, trade-off, or dependency relationships among key stakeholder interests have been recorded.

  It is *not* an exit criterion that such conflicts be resolved. Since domain engineering often involves multiple stakeholders within or across organizations, conflicts and trade-offs may occur. The representation used for the CANDIDATE PROJECT STAKEHOLDER MODEL must be able to accommodate conflicting stakeholder interests.

## Validation and Verification

- Check the list of stakeholders for completeness with respect to the fundamental domain engineering roles outlined above.

## 5.1.2 Identify Candidate Project Objectives

### Purpose

The primary goal of this task is to ensures that all relevant stakeholder interests have been considered in defining project objectives. The main function is to transform general CANDIDATE STAKEHOLDER KNOWLEDGE, and goals outlined in the PROJECT CHARTER, into specific CANDIDATE PROJECT OBJECTIVES. Closure on project stakeholders and objectives is obtained in the subsequent *Select Project Stakeholders and Objectives* task.

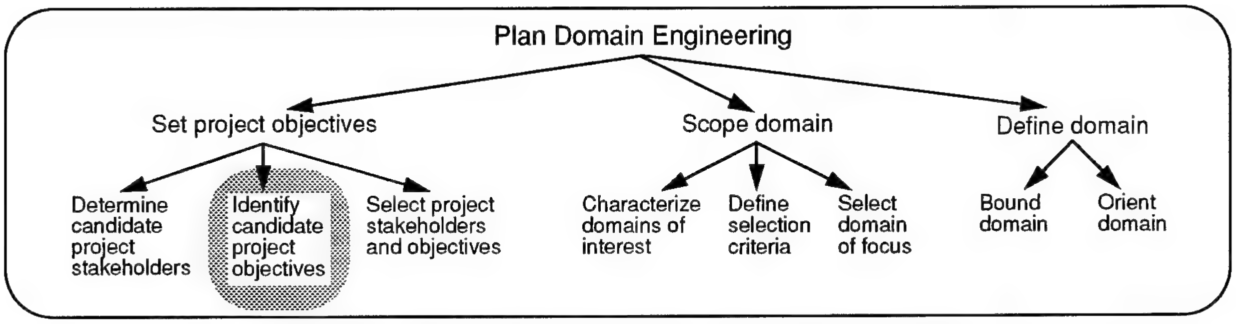

Plan Domain Engineering
Set project objectives
Scope domain
Define domain
Determine candidate project stakeholders
Identify candidate project objectives
Select project stakeholders and objectives
Characterize domains of interest
Define selection criteria
Select domain of focus
Bound domain
Orient domain

**Exhibit 16.** Identify Candidate Project Objectives Process Tree

The key challenge in this task is to determine hidden agendas, perceived opportunities, and the risk of developing project objectives contradictory to stakeholder interests. For example, if practitioners in a selected domain perceive their domain knowledge as an important element in their job security, this could radically affect both objectives and data acquisition strategy.

### Entrance Criteria

- This activity can be initiated as soon as some initial stakeholders have been identified in the *Determine Candidate Project Stakeholders* task. The CANDIDATE PROJECT STAKEHOLDER MODEL does not have to be complete. The PROJECT CHARTER must also be available to filter those stakeholder interests that are relevant to candidate objectives for the project.

### Inputs

- CANDIDATE PROJECT STAKEHOLDER MODEL. Stakeholder interests are one important source of CANDIDATE PROJECT OBJECTIVES.

- CANDIDATE STAKEHOLDER KNOWLEDGE. Supplementary information gained from stakeholders from which CANDIDATE PROJECT OBJECTIVES can be elicited.

### Controls

- PROJECT CHARTER. Used to assist in prioritize candidate objectives. The PROJECT CHARTER are also a source of candidate objectives. One way to determine these objectives is by translating the charter into a set of objectives that encompass its intent.

Note that PROJECT CONSTRAINTS are *not* a control on this task because the task involves the development of a list of *unfiltered* candidate objectives. Constraints will be used to filter the set of candidates objectives in the *Select Project Stakeholders and Objectives* task. Even constraints that affect a single candidate objective should not be used to discard an objective if the objective has a strong link to key stakeholder interests.

## Activities

➤ Set context for objectives

Determine the categories of objectives that will be considered when identifying CANDIDATE PROJECT OBJECTIVES.

In order to insure alignment of CANDIDATE PROJECT OBJECTIVES with the needs of stakeholder organizations, a hierarchical model of objective categories may be useful, spanning:

- General business objectives for the organization,
- Software engineering objectives, and
- Overall software reuse adoption objectives.

This hierarchy can be identified for each stakeholder organization. Stakeholder interests can be linked to objectives at various levels within the hierarchy.

➤ Derive candidate objectives

Drawing from the PROJECT CHARTER and characterizations of stakeholder interests in the CANDIDATE PROJECT STAKEHOLDER MODEL, identify specific candidate objectives for the project. Project members can suggest candidate objectives. The PROJECT CHARTER can be translated into a set of objectives that encompass its intent.

What project outcomes would constitute success for this stakeholder, with respect to this interest? What outcomes would be perceived as undesirable? For example, for a product division concerned with time to market, one candidate objective might be to create an asset base that would significantly reduce development time for new products in that division.

Be sure to consider both perceived incentives and risks for each stakeholder, and to consider implications for both positive and negative outcomes in each case. For example, an engineering group may perceive a mandated standard architecture as a risk because it will impose severe constraints on performance. A project that assessed the domain and judged such an architecture infeasible might be considered a positive outcome by these stakeholders. So might an outcome where a more flexible approach to a domain architecture were recommended.

➤ Map candidate objectives to stakeholder interests

Map candidate objectives to the interests of all stakeholders. Although stakeholder interests are the source of many candidate objectives, it is now necessary to explore the impact of each objective on other stakeholders. A template for this Candidate Project Objectives workproduct is shown in Exhibit C-2.

Allow for conflicting motivations for stakeholders. Certain outcomes may present both opportunities and risks for a given stakeholder. For example, an engineering project may desire to move to a standard set of components, yet be concerned that the effort to create these components will have to be funded by the project. This exploration may lead to the discovery of new relationships between objectives and stakeholders. Consider the impact of objectives derived from the PROJECT CHARTER on each stakeholder, especially those most directly affected by the charter.

➤ Map candidate objectives to PROJECT CHARTER

Map candidate objectives to the PROJECT CHARTER to determine the fit of various objectives to the overall charter. Candidate objectives are mapped to the charter in a similar manner as they were mapped to stakeholder interests in the preceding activity.

➤ Identify potential assets of interest

Build concrete pictures of how successful reuse could be performed in the project context. Key benefits of this activity are to:

- ground the Planning phase in specific, tangible outcomes,
- allow creativity thinking in support of project planning, and
- motivate the team by building a shared vision of project goals and asset users.

Brainstorm potential end uses for domain engineering products including planning products, domain models, architectures, and assets. Challenge tacit assumptions by imagining counter-scenarios. For example, the belief that the only useful outcome of domain engineering is reusable code can be challenged by determining scenarios where other domain engineering products are of value to stakeholders.

*Do not* allow this activity to gravitate too strongly towards specific implementation strategies. A good method of mitigating this risk is to brainstorm alternative implementation strategies for any given project objective and to allow these alternatives to remain as open possibilities.

## Workproducts

■ CANDIDATE PROJECT OBJECTIVES

A list of candidate objectives for the domain engineering project mapped to stakeholder interests and the PROJECT CHARTER, qualified as positive, negative, or neutral. A template for this workproduct is shown in Exhibit C-2.

Example domain engineering project objectives:

- Define standard specification models supporting domain-specific completeness and consistency checking at the requirements level.
- Reduce the total code size for a set of maintained systems, both to reduce resource requirements and maintenance burden.
- Codify and document veteran engineers' expertise in a given domain, in an organization that considers the expertise of its personnel as a key strategic asset.

## Guidelines

- Make candidate and selected objectives explicit. Teams attempting domain engineering for the first time may be tempted to perform domain selection with informal project objectives.

The need for more explicit setting of project objectives may not really become evident until later in the domain engineering process. If explicit objectives are lacking, there may be insufficient basis for selecting among candidate domains in the *Select Domain of Focus* task. If this problem occurs, backtrack to the *Identify Candidate Project Objectives* task to develop formal objectives rather than attempting to close on domain selection.

- Address non-technical objectives. Do not exclude or avoid non-technical candidate objectives. Document and consider even the politically sensitive objectives that may seem to offend the "purity" of technical criteria (if necessary, leave their source unrecorded). The primary purpose of this task is to ensure that PROJECT OBJECTIVES are realistic. Documentation of rationale for each candidate objective is secondary.
- Consider domain modelers' objectives. Domain engineering project members are project stakeholders. Consider objectives, including personal objectives, of those who will participate in the modeling effort. E.g., "performing domain analysis to learn about a new and interesting domain" is a legitimate candidate objective for a project member (although it may have lower priority than company business objectives).

### Exit Criteria

- A set of CANDIDATE PROJECT OBJECTIVES has been explicitly defined.
- Each objective is mapped to the PROJECT CHARTER and/or a key stakeholder interest.

The development of quantified project success criteria in this task would be premature. Quantified success criteria are not needed for trade-off analysis between objectives. Chosen project objective such as "develop an asset base that will improve time to market" will be quantified in subsequent tasks to provide measurable success criteria.

### Validation and Verification

- Each intuitive and informal success criterion is mapped to the stated interests of some candidate stakeholder.
- At least some objectives have been identified that are not merely reflections of external mandated requirements or constraints on the project.
- Some potentially conflicting objectives have been identified. Be wary if only consistent objectives emerge from the *Identify Candidate Project Objectives* task. The process should reveal alternatives and trade-offs; otherwise some relationships between objectives and stakeholders may not have been considered.

## 5.1.3 Select Project Stakeholders and Objectives

### Purpose

The goal of this task is to select specific project stakeholders and objectives that are consistent with the PROJECT CHARTER and attainable given PROJECT CONSTRAINTS and resources with an acceptable level of risk. The objectives selected should satisfy key stakeholder interests and maximize the chances for adoption of domain engineering results and technologies within stakeholder organizations. Once objectives and stakeholders are selected, clear project expectations should be established with all stakeholders, including project members.

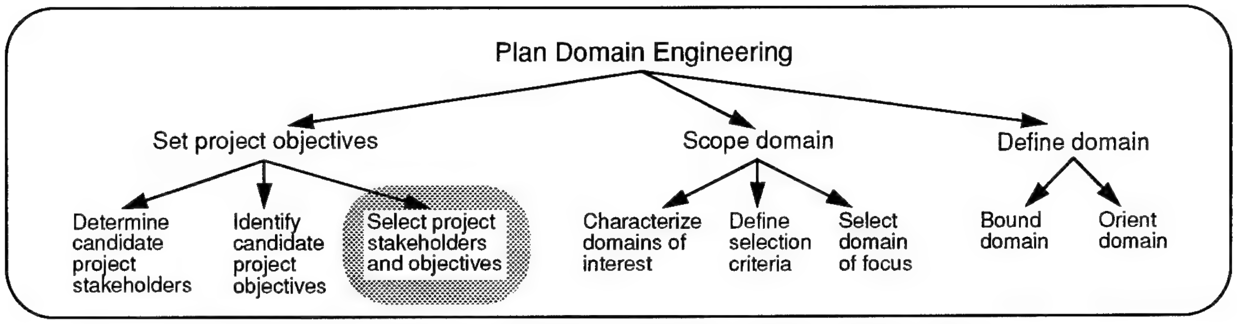

Plan Domain Engineering
Set project objectives
Scope domain
Define domain
Determine candidate project stakeholders
Identify candidate project objectives
Select project stakeholders and objectives
Characterize domains of interest
Define selection criteria
Select domain of focus
Bound domain
Orient domain

**Exhibit 17.** Select Project Stakeholders and Objectives Process Tree

## Entrance Criteria

- Since *Select Project Stakeholders and Objectives* involves trade-off analysis between stakeholder interests and objectives, both *Determine Candidate Project Stakeholders* and *Identify Candidate Project Objectives* must be completed before the *Select Project Stakeholders and Objectives* task begins.

## Inputs

- CANDIDATE PROJECT OBJECTIVES. Candidate objectives are selected as project objectives.
- CANDIDATE PROJECT STAKEHOLDER MODEL. Candidate stakeholders are selected as project stakeholders.

## Controls

- PROJECT CONSTRAINTS. Restrictions imposed by organization and market conditions beyond the project scope. Constraints may include implicit expectations about project results, perceived risks, and other factors that could compromise the success of technically sound results.

  Constraints may include the following:

  — Technology constraints

  — Constraints on domain selection

  — Available resources, as well as constraints on resources that can be tapped. For example, there may be a pragmatic constraint that the domain engineering effort cannot have any impact on the schedule or resources available for an application engineering project of high priority. Such a constraint can significantly shape viable objectives for the project.

  — Restrictions on the scope of organization boundaries that the project can cross. For example, a domain engineering project in a software maintenance organization may have restricted access to organizations that developed application systems in the domain.

  PROJECT CONSTRAINTS will be iteratively identified and documented as CANDIDATE PROJECT OBJECTIVES raise issues that reveal previously unconsidered or implicit constraints.

The PROJECT CHARTER is *not* a control to this task. Relevant information should have been incorporated into the CANDIDATE PROJECT OBJECTIVES input in the *Identify Candidate Project Objectives* task.

### Activities

➤ Assess each objective

Given available resources and other PROJECT CONSTRAINTS, assess each candidate objective in terms of:

- level of risk (how attainable is it?),
- estimated level of effort, and
- anticipated pay-off, in terms of the set of stakeholder interests most directly addressed.

➤ Perform trade-off analysis

Identify ***key stakeholders*** in the CANDIDATE PROJECT STAKEHOLDER MODEL — those stakeholders whose buy-in and commitment are deemed essential to the overall success of the project. Prioritize other stakeholders.

➤ Select the set of project objectives

Project objectives are selected from the CANDIDATE PROJECT OBJECTIVES, based on the interests of key stakeholders and other stakeholders with the highest priority. Candidate objectives that are technically valid but rejected for the current project can become potential objectives for subsequent domain engineering efforts. Objectives selected and project stakeholders whose interests are satisfied by the objectives should be documented in the PROJECT OBJECTIVES and PROJECT STAKEHOLDER MODEL.

➤ Establish success criteria and metrics

Identify success criteria with respect to particular objectives and stakeholders interested in these objectives. Define measurements and metrics that will be used to demonstrate project success. Also identify quality criteria.

Then map the success and quality criteria against both key and other stakeholders, as a cross-check for where particular incentives and barriers will apply. Use the information from previous assessments to rate each stakeholder's overall level of commitment to each objective. A template for this activity is shown in Exhibit C-3, STAKEHOLDER — OBJECTIVES MAP.

Objectives and success criteria can be further decomposed into specific metrics and evaluation criteria. These will determine aspects of the overall project plan, such as what kind of metrics data needs to be collected to demonstrate that particular objectives have been met, and when review and evaluation activities need to be scheduled to ensure that progress with respect to objectives is monitored at reasonable intervals.

For example, suppose that one objective is to estimate the level of effort that will be required for subsequent domain engineering projects in the same organization. This implies that metrics on level of effort are critical outputs of the current domain engineering project. If the current project involves creating infrastructure that will be utilized on subsequent projects, sufficient metrics need to be captured to allow this infrastructure development effort to be deducted from total effort to determine the level of effort for domain engineering.

➤ Obtain stakeholder commitment.

Since objectives are the project's contract with stakeholders in the organization, reviews of the objectives should be held with these stakeholders. At the reviews present the objectives, indicate the motivating interests, and describe risk mitigation steps that address possible concerns. Stakeholder sign-off must be obtained on both the acceptance of each objective and the *absence* of other objectives. Feedback from stakeholders is used to update the PROJECT STAKEHOLDER MODEL and PROJECT OBJECTIVES.

Buy-in from application groups that could potentially use assets produced by the domain engineering project is of primary importance. Be careful that commitment to reuse is expressed appropriately. Assets must be reused based on their own technical merits. While reuse policies and incentives showing support of management can encourage reuse, reuse mandates by management may be counter productive.

## Workproducts

■ PROJECT OBJECTIVES

A list of the selected project objectives linked to the project charter and project stakeholder interests. Objectives should be documented clearly and concisely, in language that will easily understood by new project members over the lifetime of the project. The objectives also serve as a statement of expectations for upper management.

- STAKEHOLDER — OBJECTIVES MAP: Maps objectives (success and quality criteria) against stakeholders to assess level of commitment to each objective. A template of this workproduct is illustrated in Exhibit C-3.

■ PROJECT STAKEHOLDER MODEL

The PROJECT STAKEHOLDER MODEL contains a list of the stakeholders whose interests will be satisfied by the selected PROJECT OBJECTIVES. Stakeholders are annotated with descriptions of roles, relationships, interests, enablers and barriers, and assessments of attitudes towards reuse and reuse practices. The PROJECT STAKEHOLDER MODEL contains stakeholders from the CANDIDATE PROJECT STAKEHOLDER MODEL who are still of interest to the project and any new stakeholders that emerge as a result of selected objectives. If the roles of any stakeholders will change if the domain engineering project is successful, these role changes should also be described in the PROJECT STAKEHOLDER MODEL.

## Guidelines

- Do not confuse objectives with domain choices. A typical mistake is to confuse overall domain engineering objectives with domain selection criteria. Objectives should not directly reference a specific domain choice or specific domain selection criteria, unless these constraints are imposed by the organization. Normally the domain selection criteria and domain selection should be determined in the *Scope Domain* task.

- Identify technology and business objectives. Conflicting technology and business objectives are a classic source of problems in domain engineering projects. There are typically technology advocates in an organization, both software developers and technology enthusiasts.

There are also stakeholders concerned primarily with addressing business objectives. If both objectives are selected for the project, clear priorities must be set.

- Defer technology choices when possible. Known constraints on technology decisions should be reflected in the PROJECT CHARTER or PROJECT CONSTRAINTS. Try to state domain engineering objectives in a form that doesn't unduly prescribe technology choices that are not mandated. For example, an objective to build a library of Ada code components to support the selected domain has a specific implementation approach (Ada) embedded within it. This implementation approach may or may not prove appropriate for the domain. Informed technology choices should be made during the *Architect Asset Base* and *Implement Asset Base* sub-phases, based on the characteristics of the domain. If technology commitments are made prematurely, much of the benefit of the ODM approach to domain engineering will be lost.

In some cases, valid constraints on technology selection will appear as scheduling constraints rather than a preference for a certain technology. Interdependencies between the domain engineering project and other engineering projects may create pressure to make some technology choices rapidly (e.g., the choice of architecture representation). Conversely, dependencies on technology choices of external groups may introduce some uncertainty into project planning.

## Exit Criteria

- A set of domain engineering objectives has been defined and agreed upon by the project members.
- Core customers have been identified among the stakeholders.

## Validation and Verification

- Objectives mapped to stakeholder interests. Each objective has been mapped to at least one stakeholder interest.
- Validation of objectives. Objectives are validated in terms of the following criteria:
  - Minimal overlap between objectives to ensure the list is concise and understandable.
  - Interests of key stakeholders are satisfied by some subset of objectives.
  - No project objectives are in serious conflict with interests of key stakeholders. Any direct conflicts between interests of key stakeholders and project objectives must be resolved or at least flagged as a risk. Resolving key conflicts may require involvement and decisions beyond the immediate control of the project. Less critical conflicts and trade-offs can be resolved at a later stage of the project by tuning domain engineering objectives. The goal is to get a stable set of objectives that subsequent Planning tasks can be based on.
  - The set of selected objectives constitute a sufficient guarantee of success for the project. I.e., if all objectives are satisfied, will the project be considered a success?
- Commitment obtained. Key stakeholders have seen and approved the PROJECT OBJECTIVES. Their key interests are satisfied by the selected objectives. Interests and objectives of each key stakeholder with respect to the project have been resolved for consistency. If a consistent set of key stakeholders cannot be resolved, iterate through the task again. If one or two iterations cannot resolve the impasse, it may be best to abort the project.

## 5.2 Scope Domain

### Purpose

In the previous sub-phase, *Set Project Objectives*, a set of explicit PROJECT OBJECTIVES was selected and aligned with the interests of project stakeholders. The primary purpose of the *Scope Domain* sub-phase is to make a DOMAIN SELECTION for the project that is in alignment with these PROJECT OBJECTIVES. The selected domain is called the ***domain of focus***.

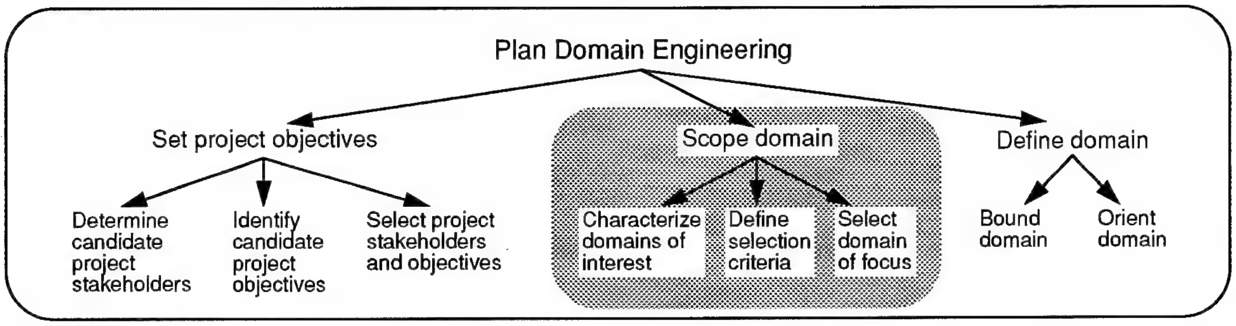

Plan Domain Engineering
Set project objectives
Scope domain
Define domain
Determine candidate project stakeholders
Identify candidate project objectives
Select project stakeholders and objectives
Characterize domains of interest
Define selection criteria
Select domain of focus
Bound domain
Orient domain

**Exhibit 18.** Scope Domain Process Tree

This term reveals something of the nature of this task. Domains are abstractions that group particular sets of systems, or areas of functionality within systems, in ways that allow strategic reuse of assets within the domain scope. Selecting a domain is not like choosing an item out of a catalog; it involves deciding where to most strategically draw this boundary. This can involve grouping systems and functional areas of interest in ways unfamiliar to stakeholders. These less obvious domain boundaries sometimes turn out to have the highest potential for payoff.

A second major objective of this sub-phase is therefore to probe possibilities for potential domains thoroughly enough that possible risks are identified, and an optimum selection is made. *Selection of the domain is the single most important strategic decision of the domain engineering project life cycle.*

### Description

As shown in Exhibit 19, the *Scope Domain* sub-phase involves three tasks:

- The *Characterize Domains of Interest* task performs a scan for possible domains, working both with the SYSTEMS OF INTEREST and with CANDIDATE STAKEHOLDER KNOWLEDGE to ensure that domains important to stakeholders are identified and considered, but *deferring* direct consideration of the PROJECT OBJECTIVES to ensure that diverse possibilities for domains within the ORGANIZATION CONTEXT have been probed as well.
- The *Define Selection Criteria* task documents the DOMAIN SELECTION CRITERIA that will drive the choice of domain, considering general REUSE CRITERIA as well as project-specific criteria aligned with the PROJECT OBJECTIVES.
- The *Select Domain of Focus* task evaluates candidates from the DOMAINS OF INTEREST according to the DOMAIN SELECTION CRITERIA. The output of this final task is the DOMAIN SELECTION and refinements to each of the primary information flows to reflect the new domain focus. These workproducts provide the information for the subsequent sub-phase, *Define Domain*, where the general area of focus selected in this sub-phase is turned into a formal bounded domain.

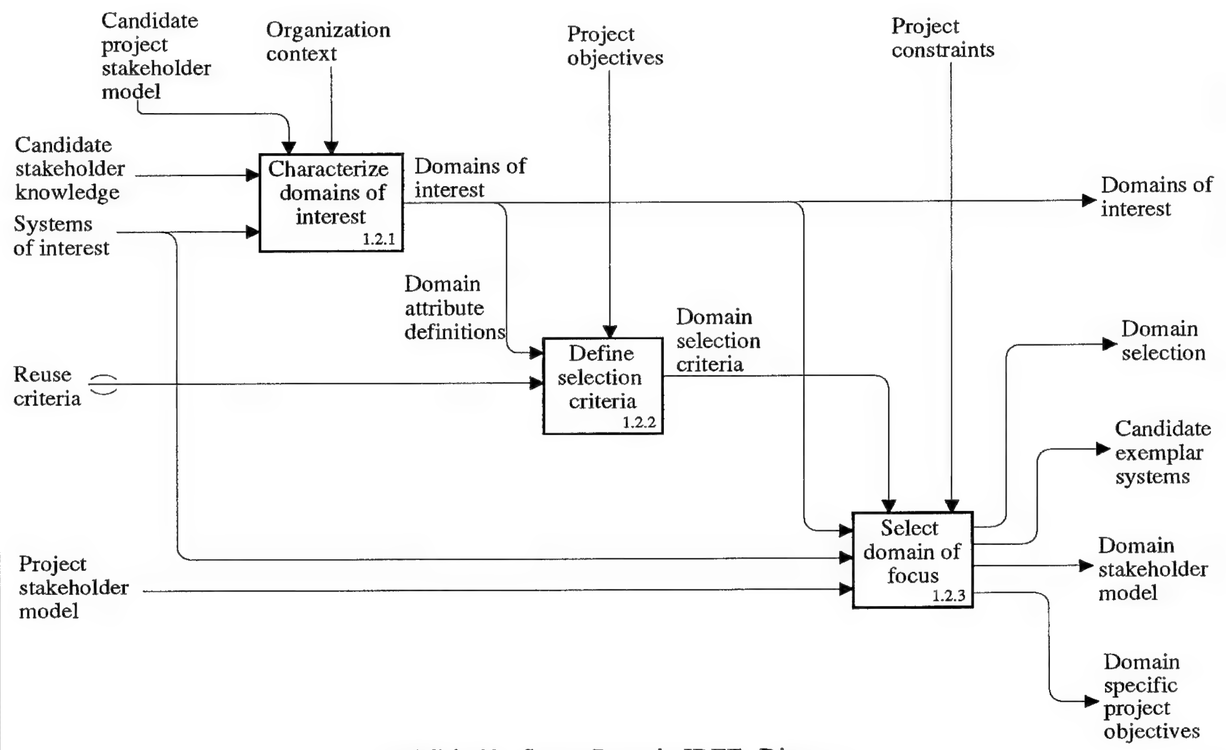

Candidate project stakeholder model
Organization context
Project objectives
Project constraints
Candidate stakeholder knowledge
Systems of interest
Characterize domains of interest 1.2.1
Domains of interest
Domains of interest
Domain attribute definitions
Domain selection criteria
Domain selection
Reuse criteria
Define selection criteria 1.2.2
Candidate exemplar systems
Select domain of focus 1.2.3
Project stakeholder model
Domain stakeholder model
Domain specific project objectives

**Exhibit 19.** Scope Domain IDEF$_0$ Diagram

This process structure helps meet the objectives of this sub-phase by separating consideration of the overall ORGANIZATION CONTEXT, the PROJECT OBJECTIVES, and the PROJECT CONSTRAINTS into three distinct tasks. Since PROJECT CONSTRAINTS are the most volatile, maximum information is retained to respond to changes in available resources, schedule, etc. in adjusting or re-working the domain focus. The structure will yield the greatest benefit when the ORGANIZATION CONTEXT stays stable, as is the case in most large organizations conducting domain engineering in a well-defined business area or line of business. In these situations, the DOMAINS OF INTEREST work-product will have intrinsic value, besides its input into the *Select Domain of Focus* task, as a "domain's-eye" perspective on the legacy systems.

## Sequencing

- The separation and sequencing of these tasks works best in situations where there is a clear business area that establishes criteria for completeness of the DOMAINS OF INTEREST list. In situations with weak ties to a coherent line of business (e.g., educational or training settings, or technology evaluation projects where the domain choice is of less significance) there may be less value in separately performing *Characterize Domains of Interest.*

- Definition of selection criteria in the *Define Selection Criteria* task and selection of a domain in the *Select Domain of Focus* task may need to be performed iteratively. In the *Select Domain of Focus* task, the domain of focus is identified based on the DOMAIN SELECTION CRITERIA. The resultant domain of focus may feel intuitively wrong, fail to get consensus from the project team, or fail to get buy-in and commitment from key stakeholders. Such breakdowns should reveal additional selection criteria and/or different weights on current selection criteria. The *Define Selection Criteria* task should be performed again to account for new selection criteria or weights. The *Select Domain of Focus* task is then performed again based on the revised DOMAIN SELECTION CRITERIA.

## 5.2.1 Characterize Domains of Interest

### Purpose

The primary goal of the *Characterize Domains of Interest* task is to perform a domain-oriented scan of the ORGANIZATION CONTEXT for the project. The main function is to identify and discover DOMAINS OF INTEREST within identified SYSTEMS OF INTEREST. A major challenge in this task is to balance identification of familiar versus innovative domains.

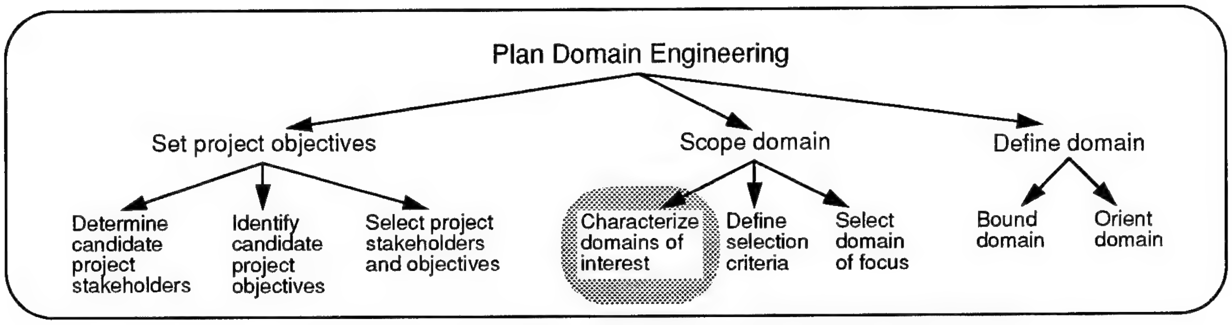

Plan Domain Engineering
Set project objectives
Scope domain
Define domain
Determine candidate project stakeholders
Identify candidate project objectives
Select project stakeholders and objectives
Characterize domains of interest
Define selection criteria
Select domain of focus
Bound domain
Orient domain

**Exhibit 20.** Characterize Domains of Interest Process Tree

Each organization has its own way of naming and organizing functional areas that may span systems. The *Characterize Domains of Interest* task allows identification of domains that are recognized and familiar to stakeholders (an ethnographic perspective). At the same time, the task allows for discovery of domains that may not be considered as functional areas within systems but are potentially good candidates for reuse, or will be otherwise useful in the domain engineering process (an innovation perspective).

Identifying a set of DOMAINS OF INTEREST before selecting a focus domain encourages exploration of alternative domains that may not be part of the informal belief model about reuse within the stakeholder organizations, but may be the best opportunities for reuse within those organizations. Exploration of these alternative domains helps to ensure selection of an optimal rather than merely an acceptable domain of focus.

### Entrance Criteria

- This task can be initiated as soon as there is a well-defined ORGANIZATION CONTEXT to bound.
- The CANDIDATE PROJECT STAKEHOLDER MODEL need not be complete, but should include an initial set of stakeholders.
- The task can be initiated before PROJECT OBJECTIVES are completely determined. The risk in beginning the task this early is that scoping criteria will be insufficient to control the task.

### Inputs

- CANDIDATE STAKEHOLDER KNOWLEDGE: Used to elicit terms used within the stakeholder community for various domains. May be obtained through:

— literature which discusses or classifies systems of interest to candidate stakeholders, e.g., survey articles or tutorials by acknowledged experts, or

— direct interviews with informants.

This input is not confined to *project* stakeholders. Since DOMAINS OF INTEREST include more than just candidates for the project's domain of focus, the intent of this task is to scan the full range of stakeholder interests.

- SYSTEMS OF INTEREST: An inventory of applications (software, systems, organizational processes) within the project scope. In a development environment, SYSTEMS OF INTEREST could include both legacy systems and anticipated new systems. In a maintenance environment, SYSTEMS OF INTEREST could include the inventory of systems under control of the group initiating the domain engineering project. SYSTEMS OF INTEREST are used to characterize DOMAINS OF INTEREST in terms of their intersection with these systems.

## Controls

- ORGANIZATION CONTEXT. Used to filter identified domains. May include contextual factors beyond those directly relevant to the PROJECT CHARTER; hence, this control is not intended to limit DOMAINS OF INTEREST to candidate domains of focus. Aspects of organization context may also be incorporated as data in workproducts, identifying the organization scope of different domains and systems of interest.
- CANDIDATE PROJECT STAKEHOLDER MODEL. Candidate project stakeholder interest provide a scoping criterion to determine which domains are of interest.

## Activities

➤ Record terms used by stakeholders for domains

Before generating the list of DOMAINS OF INTEREST, it is helpful to capture stakeholders' definitions of the term "domain". This definition establishes criteria to apply to candidate domains as they are identified.

Using interviews and written information sources determine:

- How is a domain different from a system?
- What kinds of non-standard systems in this context are termed domains (e.g., technical areas not bounded by a single contract, project or product)?

Also elicit any terms used for classes or families of systems or functional subsystems.

➤ Establish domain identification criteria.

Planners may loosely or tightly filter domains to determine of DOMAINS OF INTEREST, by specifying the criteria for inclusion in the DOMAINS OF INTEREST. DOMAINS OF INTEREST may be constrained to ensure some uniformity in terms of size, complexity, or other measures. This will simplified identification of DOMAINS OF INTEREST because domains will be more consistent in nature. The problem with this approach is that novel domains may be excluded.

➤ Identify DOMAINS OF INTEREST.

Using stakeholders' definitions and interpretations of the word "domain", elicit examples of specific domains of interest. Also consider potential domains that may not be recognized as such by stakeholders. For example, stakeholders may assume that domains must be centered around executable code rather than requirements specifications, or that firmware would not qualify as a software domain.

Because the scope or boundaries of many domains suitable for reuse are not necessarily reflected in the structure of existing systems, alternative methods for generating DOMAINS OF INTEREST can also be employed. These methods include considering unions and intersections of domains derived from stakeholder terms.

➤ Characterize DOMAINS OF INTEREST

Identify and document the relationships between domains of interest.

➤ Map DOMAINS OF INTEREST to SYSTEMS OF INTEREST

Map the intersection of DOMAINS OF INTEREST with SYSTEMS OF INTEREST. All software-based systems within the scope of candidate project stakeholders should be considered when performing this mapping. A template for this activity is shown in Exhibit C-4, DOMAIN TO SYSTEM MAP.

In ODM ***vertical domains*** refer to domains that include entire systems in their scope. These systems may be closely related (e.g., a set of system versions or a product line) or loosely related (a family of systems). A ***horizontal domain*** is a domain that includes only a subset of the functionality implemented in a system. If this functionality is clustered in one structural component of the system in question, the domain is called an ***encapsulated*** domain. Conversely a ***distributed*** domain groups functionality that is pervasive or global within a system.

In ODM the terms *vertical* and *horizontal* are not absolute qualities of general functional areas, but describe the span of a given domain's coverage relative to a particular set of systems. For example, within a family of large-scale systems, database management may be considered a horizontal domain. In the Database Management Systems provider marketplace, database management could be considered a vertical domain. Domain relationships are not analogous to system, subsystem, and module relationships. Domains can intersect, enclose other domains, be disjoint, etc. They are better thought of as set descriptions.

## Workproducts

■ DOMAINS OF INTEREST

The DOMAINS OF INTEREST workproduct includes:

- DOMAINS OF INTEREST (LIST). Annotated list of DOMAINS OF INTEREST. Useful information that cannot easily be associated with only one domain of interest should also be captured.
- DOMAIN ATTRIBUTE DEFINITIONS
- DOMAINS OF INTEREST CHARACTERIZATION. Matrix characterizing the selected set of domains of interest according to the selected attributes.
- DOMAIN TO SYSTEM MAP. Many-to-many mapping of DOMAINS OF INTEREST to SYSTEMS OF

INTEREST. A system is an ***exemplar*** of a domain if the domain functionality occurs within the system scope. The DOMAIN TO SYSTEM MAP characterizes domains by their exemplar systems. A template of this workproduct is illustrated in Exhibit C-4.

- DOMAINS OF INTEREST LEXICON. This initial lexicon contains stakeholder terms that refer to principles for grouping families of systems together, or for subsets of functionality within systems or across the life cycle of systems.

The DOMAINS OF INTEREST workproduct serves many purposes, so DOMAINS OF INTEREST should be characterized in this workproduct even if the project is constrained in its domain selection. Purposes of the DOMAINS OF INTEREST workproduct include:

- Candidate domains in the DOMAINS OF INTEREST are input to selecting the domain of focus.
- Input to the *Orient Domain* task, used in developing the DOMAIN INTERCONNECTION MODEL.
- Reused in subsequent domain engineering projects as a starting point for domain selection. If the ORGANIZATION CONTEXT is stable, the DOMAINS OF INTEREST should change relatively slowly (e.g., only if workers develop expertise in new areas, or if existing areas are partitioned into new domains).

## Guidelines

- Do not screen out DOMAINS OF INTEREST that appear to be poor candidates for application of reuse technology. DOMAINS OF INTEREST should not be screened because they are also an input to the *Orient Domain* task, where they are a source of information about domains related to the domain of focus. Screening takes place in subsequent tasks.
- DOMAINS OF INTEREST are not like systems or modules of systems. DOMAINS OF INTEREST are not necessarily disjoint. They may overlap and even enclose one another. Principles by which systems are clustered into various domains may differ widely from case to case. The level of granularity of domain functionality within systems may also vary widely. Remember that the main objective is to find a suitable domain for performing domain engineering, and each of the DOMAINS OF INTEREST may prove a potentially useful basis for reuse, even if the set of DOMAINS OF INTEREST is very diverse.

## Exit Criteria

- All stakeholder terms for domains have been mapped to some domain of interest.
- All SYSTEMS OF INTEREST to candidate project stakeholders are included in at least one vertical domain.
- Each domain of interest is mapped to at least one system of interest.

Selection of the domain of focus is *not* an exit criterion. Domain of focus selection is performed in the next task.

Interconnections between domains of interest do *not* need to be documented exhaustively.

## Validation and Verification

Since domains are not merely modules or components of systems, there are no absolute criteria for completeness of DOMAINS OF INTEREST for a given stakeholder context. In particular, while each domain should be traceable to *some* stakeholder and *some* system of interest, don't attempt to elicit all such relationships in this step. DOMAINS OF INTEREST selected can be fractal, overlapping, and infinitely refinable. More complete elicitation of these relationships is done for the domain of focus in the final task in this sub-phase, *Select Domain of Focus.*

## 5.2.2 Define Selection Criteria

### Purpose

The primary goal of the *Define Selection Criteria* task is to develop criteria to guide the *Select Domain of Focus* task. The main function is the transformation of overall PROJECT OBJECTIVES into specific DOMAIN SELECTION CRITERIA, and the filtering of general REUSE CRITERIA to include those relevant to the project context. The main challenge is to integrate and reconcile general criteria with PROJECT OBJECTIVES.

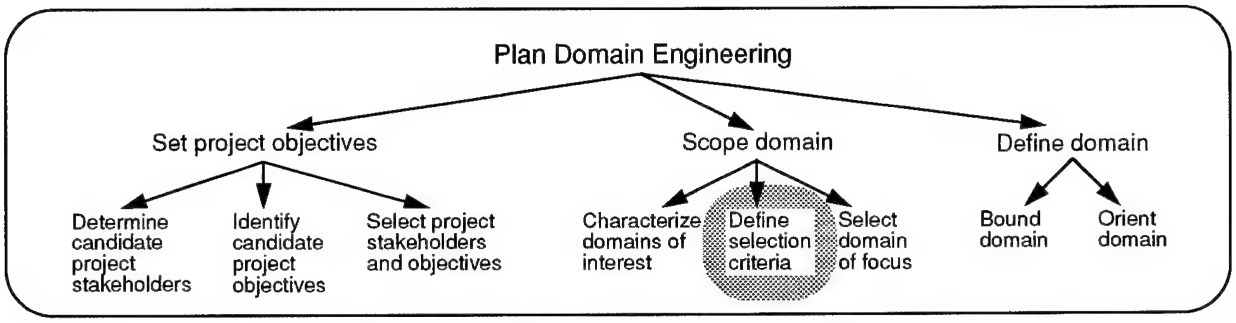

Plan Domain Engineering
Set project objectives
Scope domain
Define domain
Determine candidate project stakeholders
Identify candidate project objectives
Select project stakeholders and objectives
Characterize domains of interest
Define selection criteria
Select domain of focus
Bound domain
Orient domain

**Exhibit 21.** Define Selection Criteria Process Tree

Criteria for domain selection are determined separately from the *Characterize Domains of Interest* task because there is value in characterizing domains that would clearly not be good candidates for domain engineering based on the project context. For example, a company involved in data-intensive applications should carefully consider the evolving DBMS domain, although the company may not have such a domain within its own engineering scope.

Criteria for identifying good domains for reuse are also determined separately from domain selection in the *Select Domain of Focus* task. The same criteria can be useful for many iteration of domain selection. These iterations of domain selection can take place when resources expand or contract. While criteria may include constraints on the domain of focus such as availability of domain experts, the current life cycle phase of likely recipient application projects, etc., actual PROJECT CONSTRAINTS are addressed in the *Select Domain of Focus* task.

Documenting DOMAIN SELECTION CRITERIA helps to reveal implicit conflicts between organization beliefs and general REUSE CRITERIA prior to domain selection. It can also provide a means of allowing upper management to sign-off on the method used for domain of focus selection, while leaving the actual selection task under the control of the domain engineering project members.

### Entrance Criteria

- Selection criteria based on general REUSE CRITERIA can be derived at any time.
- Selection criteria based on a domain attribute definition can be derived as soon as the attribute definition is available.

- Selection criteria based on a domain of interest can be derived as soon as the domain is characterized.

## Inputs

- <u>DOMAIN ATTRIBUTE DEFINITIONS</u>. One source of DOMAIN SELECTION CRITERIA. Not all identified attributes of DOMAINS OF INTEREST are relevant to the selection of the domain of focus.
- <u>REUSE CRITERIA</u>. General domain-independent criteria for selecting promising domains for reuse. Used here both as a source of DOMAIN SELECTION CRITERIA, and as a cross-check for project-specific criteria derived from PROJECT OBJECTIVES.

## Controls

- <u>PROJECT OBJECTIVES</u>. Used as the primary means of filtering selection criteria into those appropriate for the project. Also one input used to determine DOMAIN SELECTION CRITERIA.

## Activities

➤ <u>Derive criteria from PROJECT OBJECTIVES</u>

Working from the PROJECT OBJECTIVES, derive criteria that will be used for selection of the domain of focus. Each objective may yield several criteria. Assemble criteria independently for each objective. If criteria have an implied weighting or preference, the preference should be recorded.

For example, a project objective could be to "validate technology for developing flexible asset base architectures". This objective suggest that the degree of structural variability would be a significant criterion. A distributed horizontal domain would tend to exhibit greater structural variability across exemplar systems. Degree of structural variability would be added to the criteria list, with a preference for greater as opposed to less variability indicated as a weighting.

➤ <u>Derive criteria from DOMAIN ATTRIBUTE DEFINITIONS</u>

Derive selection criteria from the DOMAIN ATTRIBUTE DEFINITIONS for the DOMAINS OF INTEREST.

➤ <u>Select relevant general REUSE CRITERIA</u>

Select those general REUSE CRITERIA for domain selection that appear relevant in the project context. Capture rationale for general objectives not considered relevant.

The following list contains some of the general REUSE CRITERIA for domain selection that can be used as a starting point for developing criteria on a particular project:

- Maturity of the domain including number of systems implemented and length of time fielded.
- Potential for future payoff. How many systems are to be implemented in the future?
- Stability of underlying technology within the domain. Does the domain encapsulate technical approaches that are due to become obsolete shortly?

- Availability of codified knowledge in the domain.
- Experience in the domain held by the organization performing the domain engineering project. Is this organization a developer of systems in the domain? A maintainer? A user?
- Estimated commonality of functions across systems in the domain.
- Degree to which performance constraints may inhibit reuse of components with excess functionality.

➤ Merge selection criteria

Criteria derived from the various sources are merged into an annotated list that includes derivation of each criterion. Criteria can conflict and even be mutually exclusive at this point.

➤ Map criteria against PROJECT OBJECTIVES

All selection criteria are mapped back to the PROJECT OBJECTIVES. The output is a matrix with objectives in the rows, mapped against criteria in the columns. Cells can be labelled "supporting", "conflicting" or "neutral".

➤ Prioritize criteria

The matrix is used to derive the final annotated list of DOMAIN SELECTION CRITERIA. If the list is a subset of the original criteria, rationale should be recorded for rejected criteria. Some criteria may be difficult to interpret as either positive or negative indicators for the selection task.

## Workproducts

■ DOMAIN SELECTION CRITERIA

A list of prioritized selection criteria that will be used to select the domain of focus, with annotations explaining the derivation of each criterion.

> *Example.* One domain selection criterion for the Army/Unisys STARS Demonstration Project [ADER95a] was that the domain selected should cover functionality contained in software on the main computer used by the exemplar systems, rather than firmware-embedded functionality on peripheral equipment. This criterion was a project-specific criterion linked to the PROJECT OBJECTIVES, not a general reuse criterion for domain selection.
>
> As a counter example, for some Hewlett Packard application groups, reuse within firmware domains is entirely appropriate to their business area.

## Guidelines

- Criteria reflect ORGANIZATION CONTEXT. Criteria should reflect as much as possible the particular ORGANIZATION CONTEXT in which the selection is being made. No criterion is wrong if it reflects a contextual constraint. Do not ignore situation-specific constraints that may seem to obscure the purity of technical criteria.
- Do not ignore general REUSE CRITERIA. Project-specific criteria may override general REUSE CRITERIA in some situations. However, general criteria cannot be ignored. For example, if a compelling reason cannot be made for reuse of assets across multiple systems in a given busi-

ness area, general REUSE CRITERIA would say that domain engineering should not be performed on this domain. However, the organization may have a desire to devote resources to software process improvement in this domain.

- Examine domain selection mandates. If the PROJECT OBJECTIVES contain assumptions about which domain is selected for domain engineering, these assumptions should be examined should be included in selection criteria. Other selection criteria may contradict project domain selection assumptions. If this is the case, the selection criteria must be resolved. If resolution is impossible, the project may have to terminate. Although termination of the domain engineering project is undesirable, it is better to terminate than to than pursue domain engineering on a domain which is a poor risk.

### Exit Criteria

- Criteria aligned with PROJECT OBJECTIVES. Criteria definition is complete when the prioritized list of DOMAIN SELECTION CRITERIA has been generated and alignment of each criterion on the list to PROJECT OBJECTIVES has been demonstrated.

### Validation and Verification

- One challenge involved in validating Domain Selection Criteria is that it is possible to derive conflicting criteria from a consistent set of PROJECT OBJECTIVES.

## 5.2.3 Select Domain of Focus

### Purpose

The primary goal of the *Select Domain of Focus* task is to select a domain of focus for the domain engineering project that satisfies PROJECT OBJECTIVES and is feasible given PROJECT CONSTRAINTS, including PROJECT RESOURCES. The main function is to make a DOMAIN SELECTION from the DOMAINS OF INTEREST, using the DOMAIN SELECTION CRITERIA. As a subsidiary function, various workproducts are updated to reflect the commitment to a specific domain. These include:

- The PROJECT OBJECTIVES are augmented with DOMAIN SPECIFIC PROJECT OBJECTIVES,
- CANDIDATE EXEMPLAR SYSTEMS for the selected domain are extracted from the SYSTEMS OF INTEREST.
- The DOMAIN STAKEHOLDER MODEL extracted from the PROJECT STAKEHOLDER MODEL by choosing stakeholders who are interested in the selected domain.

The main challenge of this task is to pick the right domain. This decision has the greatest single impact on a successful domain engineering project outcome. Poor domain modeling using the right domain can be repaired with rework. The best modeling expertise in the world, applied to a domain that is a poor strategic choice, will have little chance of success.

### Entrance Criteria

- It is possible to begin identifying candidates for the domain of focus as soon as some DOMAINS OF INTEREST have been identified in the *Characterize Domains of Interest* task.

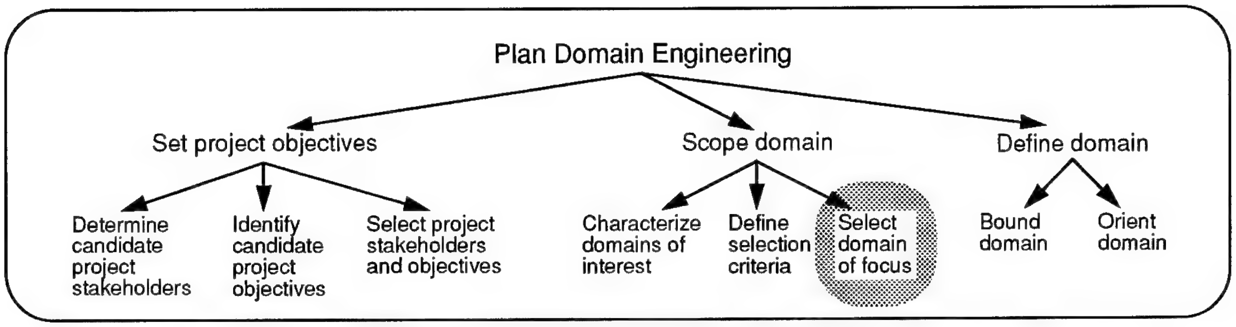

Plan Domain Engineering
Set project objectives
Scope domain
Define domain
Determine candidate project stakeholders
Identify candidate project objectives
Select project stakeholders and objectives
Characterize domains of interest
Define selection criteria
Select domain of focus
Bound domain
Orient domain

**Exhibit 22.** Select Domain of Focus Process Tree

- Detailed trade off analysis should not begin until the DOMAIN SELECTION CRITERIA have been finalized.

## Inputs

- DOMAINS OF INTEREST. Used as one source for candidates for the domain of focus. Not all candidates need to be drawn from this list, since identified DOMAIN SELECTION CRITERIA may lead to other candidates.
- SYSTEMS OF INTEREST. Used to determine the strategic value of candidate domains under consideration, in terms of the set of systems they intersect.
- PROJECT STAKEHOLDER MODEL. Refined in this task into the DOMAIN STAKEHOLDER MODEL, which includes stakeholders with interest in the selected domain. Not all project stakeholders will have a stake in the results of the project which are specific to the domain of focus.

## Controls

- DOMAIN SELECTION CRITERIA. Control the selection of the domain of focus. As task documentation, these criteria are refined based on iterative identification of criteria in this task and incorporated into the DOMAIN SELECTION report.
- PROJECT CONSTRAINTS. Some constraints have already been factored into PROJECT OBJECTIVES and the PROJECT STAKEHOLDER MODEL. In selecting the domain of focus more detailed constraints may become relevant. For example, PROJECT RESOURCES, schedules, available personnel, etc. must be considered carefully in choosing a feasible domain scope.

PROJECT OBJECTIVES are *not* a control on this task. The DOMAIN SELECTION CRITERIA should have incorporated any PROJECT OBJECTIVES relevant to the domain selection task.

## Activities

➤ Derive candidate domains of focus

Examine the DOMAINS OF INTEREST for viable candidate domains of focus. In addition, the SYSTEMS OF INTEREST and other aspects of ORGANIZATION CONTEXT can be reexamined for viable candidate domains of focus, using the DOMAIN SELECTION CRITERIA as a filter.

➤ Characterize candidate domains

Characterize the candidates with respect to the identified DOMAIN SELECTION CRITERIA. A tempe for the resulting CANDIDATE DOMAINS MATRIX is shown in Exhibit C-5.

➤ Prioritize candidate domains

Based on the weight assigned to each of the DOMAIN SELECTION CRITERIA and the characterization of the candidate domains, produce an ordered list of candidates.

➤ Annotate candidate domain list

Examine the list and note other benefits and risks of selecting particular domains. These will be unique factors that were not listed in the DOMAIN SELECTION CRITERIA. For example, the selection of a certain domain might present an opportunity for one of the domain asset users to be a new development project.

➤ Factor in PROJECT CONSTRAINTS

The DOMAIN SELECTION CRITERIA should produce a list of viable candidate domains. PROJECT RESOURCES and other PROJECT CONSTRAINTS provide a further filter for both selecting a domain, and on the overall level of effort required by the domain scope.

➤ Select the domain of focus

Using all the information obtained in the preceding activities, make the final DOMAIN SELECTION.

Obtain buy-in from the entire project team on the domain selected. Not all domain engineering decisions can be made on a collaborative, consensus basis. However, the choice of domain has pervasive impact on the entire project. Any disaffected parties with regard to this decision may cause difficulties later in the project.

➤ Update and refine previous workproducts

- Refine PROJECT STAKEHOLDER MODEL into DOMAIN STAKEHOLDER MODEL. Domain Stakeholders are selected from the Project Stakeholders by determining potential customers, based on the domain selected.
- Determine DOMAIN SPECIFIC PROJECT OBJECTIVES.
- Refine SYSTEMS OF INTEREST into CANDIDATE EXEMPLAR SYSTEMS. Based on the informal domain definition, select the CANDIDATE EXEMPLAR SYSTEMS for the domain from the SYSTEMS OF INTEREST.
- Document selection decision and rationale. Rationale includes characterization of the selected domain in terms of each selection criterion, including any advantages and risks.

## Workproducts

■ DOMAIN SELECTION (REPORT)

Documents the activities that lead to selection of a domain of focus. The DOMAIN SELECTION includes:

- The CANDIDATE DOMAINS MATRIX, which characterizes candidate domains by DOMAIN SELECTION CRITERIA. A template for this workproduct is shown in Exhibit C-5.
- A description of the SELECTED DOMAIN(S) of focus.
- An INFORMAL DOMAIN DEFINITION STATEMENT for domain of focus.

■ CANDIDATE EXEMPLAR SYSTEMS

Indicates SYSTEMS OF INTEREST that intersect the domain of focus, characterized with respect to the domain of focus. Since the domain boundary has not been fully defined, this list will be further refined in the *Bound Domain* task.

■ DOMAIN-SPECIFIC PROJECT OBJECTIVES

Contains specific objectives for the project related to the domain of focus. Also contains potential assets of interest for the domain and potential asset base customer contexts.

■ DOMAIN STAKEHOLDER MODEL

A refinement of the PROJECT STAKEHOLDER MODEL which only includes stakeholders with an interest in the domain of focus. The DOMAIN STAKEHOLDER MODEL can also include new stakeholders with interest in the domain, based on the INFORMAL DOMAIN DEFINITION STATEMENT. The DOMAIN STAKEHOLDER MODEL also includes the specific roles of these stakeholders with respect to the domain of focus.

## Guidelines

- Multi-Domain Scope. The ODM process model assumes the context of a single domain engineering project. This single project may choose a scope that includes several domains of interest. These domains of interest may result from composing or clustering several smaller domains, or through tactical selection of multiple, unrelated domains. If multiple, unrelated domains are chosen, the project should be split into separate teams that perform domain engineering on each domain.
- Document the real rationale. The activities in the *Select Domain of Focus* task imply that selection of a domain of focus should occur based on priorities of DOMAIN SELECTION CRITERIA. However, in practice the final choice of a domain of focus may not be made solely on the basis of these priorities. Document the real concerns and rationale that motivated the DOMAIN SELECTION as much as possible. Remember that planners of subsequent domain engineering projects will need to refer to this rationale as a resource.

## Exit Criteria

- The selected domain of focus matches all essential DOMAIN SELECTION CRITERIA.
- Rationale for the DOMAIN SELECTION has been documented, especially if the DOMAIN SELECTION CRITERIA weights do not fully motivate the choice.
- There is consensus among the modeling team for the choice. The selection of a domain of focus is one task where a team-based modeling approach is essential to project success.

## Validation and Verification

- Obtain commitment from stakeholders. Necessary commitment from stakeholders includes:
  - Permission to proceed with domain engineering.
  - Commitment of resources necessary to perform domain engineering on the selected domain
  - Backing to carry out the project. This includes permission to cross certain organization boundaries (e.g., talk with customers and system developers), talk to people with expertise in the domain, access documentation, etc.

## 5.3 Define Domain

### Purpose

The Planning phase up until Domain Selection has worked largely from an informal, intuitive understanding of the domain of focus. In addition, it has been a "narrowing" process: from the interests of various stakeholders to a specific set of project objectives, to a set of candidate domains that are narrowed to a selected domain of focus for the project.

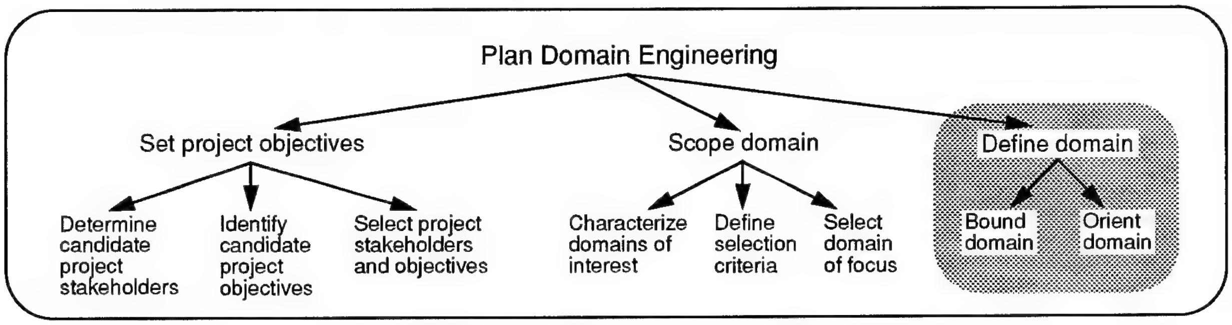

Plan Domain Engineering
Set project objectives
Scope domain
Define domain
Determine candidate project stakeholders
Identify candidate project objectives
Select project stakeholders and objectives
Characterize domains of interest
Define selection criteria
Select domain of focus
Bound domain
Orient domain

**Exhibit 23.** Define Domain Process Tree

The *Define Domain* sub-phase has a different character and dynamic. In this step we formalize this intuitive notion into a verifiable, bounded set of principles and examples. We formally define what is entailed by the selected domain, and clarify what is in and out of the domain. The domain is defined both formally, using rules of inclusion and exclusion documented in the INTENSIONAL DOMAIN DEFINITION, and empirically, by referring to application instances of the domain in the EXEMPLAR SYSTEMS SELECTION. We also "orient" the domain in terms of its historical context, broader and narrower domains, related (or "neighboring") domains, and various application contexts in which it appears; these relationships are documented in the DOMAIN INTERCONNECTION MODEL.

These definition tasks constitute a first step into domain modeling proper. Discovery of the natural boundaries of the domain of focus informs the decision whether to continue on to the full *Model Domain* phase or to iterate the *Plan Domain Engineering* phase to derive a more focused domain.

### Description

As shown in Exhibit 24, this sub-phase consists of two main tasks.

In *Bound Domain*, the definition of the domain is formalized, both in terms of rules of inclusion and exclusion and in terms of identified member systems, or ***exemplar systems***, from which a set of ***representative systems*** is selected for detailed analysis. In *Orient Domain*, a DOMAIN INTERCONNECTION MODEL is created that documents qualitative relations between the domain of focus and structurally or conceptually related domains.

The DOMAIN DEFINITION, refined and updated throughout the domain engineering life cycle to represent the actual scope of the final asset base, may be used by:

- domain planners who wish to understand the potential overlap between distinct domains;
- component developers and asset managers;
- potential utilizers, to determine whether a component they seek belongs within the scope of

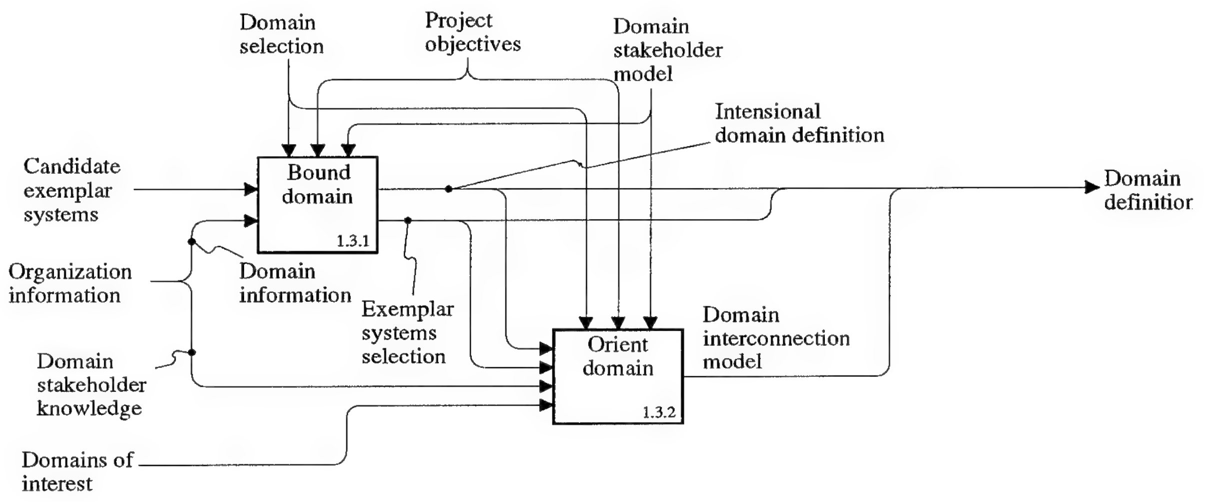


**Exhibit 24.** Define Domain IDEF$_0$ Diagram

the domain

- Submitters of draft components to the asset base, to distinguish assets absent due to deliberate exclusion from those absent due to gaps in coverage that could properly be addressed with a new asset.

### Sequencing

As with most sub-phases, these main tasks can be done sequentially or in parallel. There will be a natural tendency to interleave the activities of defining the domain boundary and qualifying related domains beyond that boundary. So, for example, when a boundary decision is made in the defining rules for the domain, this may lead to an exploration of related domains relevant to that boundary.

## 5.3.1 Bound Domain

### Purpose

During the *Select Domain of Focus* task, the domain has been defined informally. The primary goal of the *Bound Domain* task is to refine the informal definition of the domain into a set of explicit criteria for defining what systems and functions are in or out of the domain.

The primary functions performed in bounding the domain are to define the domain both in terms of rules of inclusion and exclusion of systems with respect to the domain, and by designating a set

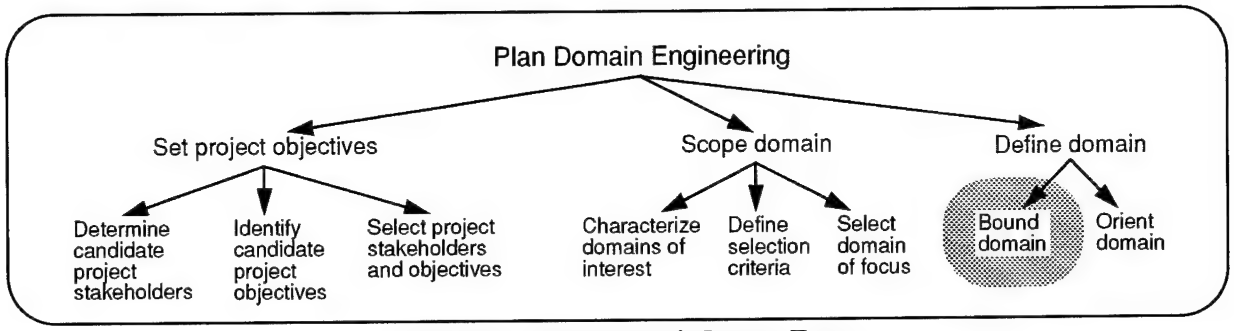

Plan Domain Engineering
Set project objectives
Scope domain
Define domain
Determine candidate project stakeholders
Identify candidate project objectives
Select project stakeholders and objectives
Characterize domains of interest
Define selection criteria
Select domain of focus
Bound domain
Orient domain

**Exhibit 25.** Bound Domain Process Tree

of systems as examples of the domain (called ***exemplar systems***). The approach to defining a domain in terms of inclusion and exclusion rules is called *intensional definition.* ***Intensional*** is a term from the field of logic which means that the rules concern essential and not accidental features of the domain; i.e., rules developed during intensional definition can be used to classify new candidate exemplar systems as being either inside or outside the domain.

The definition in terms of exemplar systems can provide empirical validation that there is shared understanding of the intensional definitions and may also be used to derive bounding rules included in the intensional definition. The exemplar systems also help to bound later Domain Modeling tasks. Articulating appropriate domain boundaries is a key challenge in domain engineering.

## Entrance Criteria

- The project team has selected the domain of focus and created the INFORMAL DOMAIN DEFINITION STATEMENT.
- Project stakeholders have given initial commitment to the selected domain.

## Inputs

- CANDIDATE EXEMPLAR SYSTEMS. Used here to develop the exemplar system selection.
- DOMAIN INFORMATION. Information about the domain known by the domain informants. Used to identify domain terminology relevant to definition of the domain. DOMAIN INFORMATION is gathered informally at this stage.

## Controls

- PROJECT OBJECTIVES. These are needed as a reference when boundary issues arise. Reference back to overall project objectives helps keep the bounding task focused on the definition of the domain that is most useful in achieving PROJECT OBJECTIVES.
- DOMAIN SELECTION. The INFORMAL DOMAIN DEFINITION STATEMENT is used as a basis for the defining rules of inclusion and exclusion for the domain. Any documented linkage to specific systems that formed part of the rationale for the domain selection also becomes input to determining the EXEMPLAR SYSTEMS SELECTION.
- DOMAIN STAKEHOLDER MODEL. Assists in the identification of exemplar systems based on the links those systems have with stakeholder organizations. Some domain boundaries may

be defined in organizational terms (e.g., "the set of all inventory control systems maintained by Division X of Company Y.")

### Activities

➤ Identify exemplar systems by roles

The following informal typology may be useful as a starting point for brainstorming candidate exemplar systems for the domain. This typology is intended to elicit domain boundary conditions in both historical and functional terms.

- Pioneer. What system is generally accepted as the earliest occurrence of the domain functionality? For example, in the Outlining domain, ThinkTank was an early idea processor generally credited with introducing the concept of interactive outline editing.
- Kernel. What is the smallest, minimum-functionality system implementing the domain functionality that could reasonably be termed an example of the domain? These exemplar systems might typically appear as embedded functions within larger system. In the Outlining domain, an outline-oriented directory browser might represent the smallest core of features reasonably classified as outlining capability.
- Model-T. What system is accepted as a basic version of domain functionality? In the Outlining domain, Acta Advantage was an early product that provided stand-alone outlining capabilities without extensive word-processing functions.
- Classic. What is the standard example of a system implementing the domain functionality? In the Outlining domain, the outline mode embedded in Microsoft Word's text editor might be considered the standard point of reference for outlining capabilities.
- Cadillac. What is considered the high-end example system for this domain? In the Outlining domain, outliners embedded in spreadsheets or presentation managers currently have the richest set of features.
- Swiss Army knife. Is there an example system that allows end-users maximum flexibility in tailoring or adapting domain functionality. In the Outlining domain, outlining capabilities within systems that have user-extensible macro languages are an example.
- Buck Rogers. An example system that seems to stretch the domain as far as possible without leaving the defined boundary. In the Outlining domain, certain hypertext systems provide support for richer data structures than strictly hierarchical outlines. Yet many of these systems still preserve the essential features of the outline domain.

➤ Identify lexicon terms

Identify and define terms to be included in the DOMAIN DEFINITION LEXICON. Include terms used for:

- defining features of the domain
- domain stakeholders
- exemplar systems
- related domains

➤ Convert informal definition into feature statements

Using the INFORMAL DOMAIN DEFINITION STATEMENT, highlight individual statements that appear to correspond to defining features of the domain. Statements about the purpose of the domain will suggest at least some high-level functionality provided to some stakeholders. These purpose statements can then be scanned for terms that have special meaning in the context of the domain.

> *Example.* In the Outlining domain, a general purpose statement might be, "Outline processors allow writers to manipulate text in units structured according to hierarchical outlines." In this statement, "text" does not appear to have a domain-specific meaning, whereas "text units" might become a high-level umbrella term for different levels of structure of the outline.

Derive DOMAIN DEFINING FEATURES by determining for each CANDIDATE EXEMPLAR SYSTEM what makes the system belong to the domain.

> *Example.* In the outlining domain, if all exemplars exhibit the capability of collapsing sub-headings, then this feature, at this level of description, could be considered a defining feature for the domain. On the other hand, if some programs did not have this capability it would become a feature of comparatively greater interest.

➤ Derive defining features

A ***feature*** is a statement that can be true or not true of an exemplar system. ***Defining features*** are statements hold for any exemplar system in the domain. If necessary, sentences in the INFORMAL DOMAIN DEFINITION STATEMENT can be split into multiple features. ***Differentiating features*** are those features that hold for only some exemplar systems. In some cases, a high-level defining feature can be described only implicitly, as the aggregate of several differentiating features.

➤ Map defining features to exemplar systems

A template for this activity is shown in Exhibit C-6, DOMAIN DEFINING FEATURES. The result of this activity is a classification of the candidate exemplars as core, counter-exemplars, or borderline exemplars. Adjustment of the defining rules, or of the candidate set of systems, may be required in this activity.

## Workproducts

■ INTENSIONAL DOMAIN DEFINITION

- DOMAIN DEFINITION STATEMENT. Includes the name of the domain. The name should be enriched with each decision about domain bounding criteria, although it may not be possible to convey all the criteria adequately in the name. The DOMAIN DEFINITION STATEMENT serves the project team as a consensus document about what the domain includes. In addition, the DOMAIN DEFINITION STATEMENT communicates to domain practitioners the specific scope addressed by the domain of focus. The definition can be targeted to experts or novices in the domain.
- DOMAIN DEFINITION LEXICON. Contains a subset of terms from DOMAINS OF INTEREST LEXICON relevant to the selected domain. Also contains terms elicited from domain stakeholders and information sources that describe defining features for the domain.
- DOMAIN DEFINING FEATURES. Contains the defining features of the domain in terms of rules

of inclusion and exclusion and the logical relationship between these rules. A ***defining rule*** is a statement about various combinations of features that imply membership in the domain.

■ EXEMPLAR SYSTEMS SELECTION

A list of exemplar systems for the domain. This list complements the *intensional* rules contained in the DOMAIN DEFINING FEATURES. The EXEMPLAR SYSTEMS SELECTION defines the domain by identifying systems judged to be members of the domain. The EXEMPLAR SYSTEM SELECTION should document *all* systems that planners have classified with respect to domain membership. Classifying a system as an exemplar does not constitute a plan to study it in detail. The set of exemplar systems to be studied intensively will be selected in the *Plan Data Acquisition* task.

The CANDIDATE EXEMPLAR SYSTEMS are placed in the following categories, reflecting degrees of membership in the domain:

- Exemplars. Systems with sufficient features for membership in the domain according to the defining rules. These will generally include a narrower set that can be termed core exemplars. ***Core exemplars*** are systems that are felt to strongly define or characterize the essence of the domain (e.g., an established industry lead product). Features identified in core exemplars may be migrated into the defining rules for the domain.
- Borderline exemplars. Systems that have features that provide evidence both for membership and non-membership in the domain, but not sufficient evident for either. Often borderline exemplars will be the focus of raised issues or disagreements within the modeling team. These conflicts are symptomatic of a domain boundary issue that may be iteratively explored and resolved via adjustment of the defining features and exemplar set.
- Counter-exemplars. Systems with features identified that provide sufficient evidence of non-membership in the domain. Since listing all counter-exemplars would be in theory an infinite task, the point is to look for systems that lie just outside the border of the domain. The most interesting counter-exemplars are those that share many defining features with the domain and have or only a few features matching rules of exclusion. Chosen in this way, each counter-exemplar serves to validate the necessity and/or sufficiency of various defining features. Another reason to document counter-exemplars is to help subsequent users of domain modeling results distinguish systems never considered for inclusion in the exemplar systems by the project team from those considered and rejected for specific reasons.
- DOMAIN DEFINING FEATURES. Documents the decision procedure and supporting rationale for classification of exemplars. A template of this workproduct is shown in Exhibit C-6.

## Guidelines

- Do not systematically elicit differentiating features for exemplar systems at this stage. In reconciling the DOMAIN DEFINITION STATEMENT rules of inclusion and exclusion with the EXEMPLAR SYSTEMS SELECTION, do not enumerate differentiating features in order to create a defining feature. For example, planners may decide that the only obvious way to create a rule of inclusion is to state that a domain "includes algorithms A, B, and C, but not D, E or F". In this examples the defining feature should be "uses algorithm A, B, or C".
- Iterate between intensional and extensional definition. In practice, iteration should occur between developing the intensional definition, i.e., rules of inclusion and exclusion in the domain, and determining the exemplar systems for the domain. These activities can be performed largely in parallel. For example, two independent groups could develop the domain definition rules and EXEMPLAR SYSTEMS SELECTION, iteratively determining that the result-

ant workproducts are compatible.

- Look for close counter-exemplars. Counter-exemplars that lie "just outside the border of the domain" are the most useful. This can be quantified in terms of the proportion of matched to unmatched features in the candidate exemplar's profile. If possible, find counter-exemplars that match all but one defining feature, yet are still *not* in the domain.

### Exit Criteria

- The domain defining features appear necessary and sufficient to characterize whether or not given systems are within the domain.
- All members of the EXEMPLAR SYSTEMS SELECTION satisfy the domain defining features.
- There is at least one exemplar system identified in the EXEMPLAR SYSTEMS SELECTION for every domain defining feature.

### Validation and Verification

- Validate domain defining features classification of candidate exemplar systems. At the end of the *Bound Domain* task, these should be consistent. Every system in the EXEMPLAR SYSTEMS SELECTION has been characterized in terms of defining features, and the features match those that allow (if not require) inclusion in the domain.
- Domain defining features should be relatively independent. Later in domain engineering, each defining feature may become the root of a feature model. Relatively independent domain defining features will facilitate this later task.

## 5.3.2 Orient Domain

### Purpose

A fundamental principle in domain modeling is that a boundary cannot be understood without knowing what is on both sides of that boundary. In the previous task, *Bound Domain*, planners developed principles of inclusion and exclusion for the domain, some or all of which were illustrated by exemplar or counter-exemplar systems. In the *Orient Domain* task, planners revisit and qualify the various boundary criteria established for the domain of focus, assign other domain names to bordering areas of functionality and classify types of relations that hold between the domain of focus and related domains.

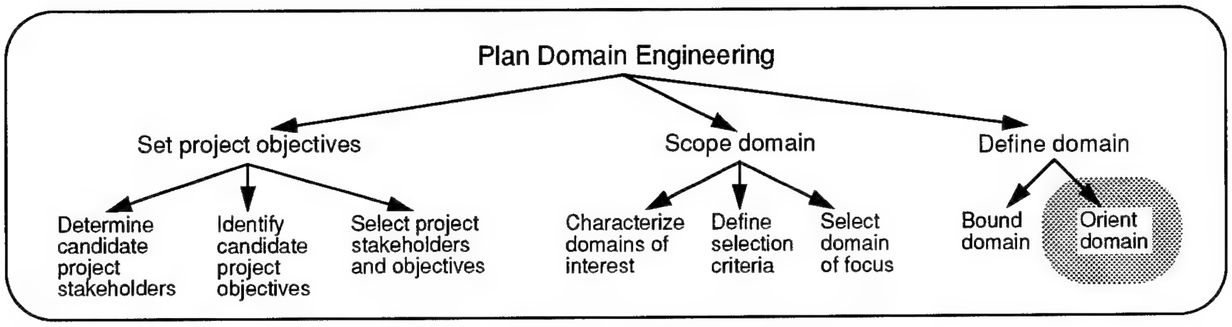

Plan Domain Engineering
Set project objectives
Scope domain
Define domain
Determine candidate project stakeholders
Identify candidate project objectives
Select project stakeholders and objectives
Characterize domains of interest
Define selection criteria
Select domain of focus
Bound domain
Orient domain

**Exhibit 26.** Orient Domain Process Tree

Besides further validating the Domain Defining Features, the *Orient Domain* task provides a means of systematically determining technical areas within the defined domain boundary for which data acquisition and modeling may not make sense. For example, in the Outlining domain, text editing functions are intimately interleaved into the operations of creating headings and sub-headings. Yet it would be futile to attempt detailed modeling of all text editing functions as part of domain modeling of outlining capabilities.

Finally, *Orient Domain* offers an early opportunity for innovation. Where bounding the domain involved a narrowing of focus on a particular domain boundary, orienting the domain involves a complementary widening that mirrors this previous narrowing task while retaining and further refining the domain boundary. In checking and validating the formal definition, modelers explore alternative contexts in which domain functionality may be applicable. Identified domain relations may eventually be reflected in layers of the asset base architecture, supporting different variants of the system or subsystem implementing the domain as a whole that can be obtained separately from the asset base.

## Entrance Criteria

- A domain of focus has been selected. There may be a temptation to begin orienting the domain directly from the DOMAINS OF INTEREST. However the coherence of domain orientation depends on considering only those domains peripheral to an identified central domain of focus (the selected domain), rather than trying to identify all domain interrelationships within the set of DOMAINS OF INTEREST.
- The domain has been provisionally bounded. Initial INTENSIONAL DOMAIN DEFINITION and EXEMPLAR SYSTEMS SELECTION have been developed. While iteration will be natural between the *Orient Domain* task and its predecessor task, *Bound Domain*, it is easier to begin domain orientation using initial workproducts developed in *Bound Domain*.

## Inputs

- INTENSIONAL DOMAIN DEFINITION. Source of DOMAIN DEFINING FEATURES, used to suggest related domains.
- EXEMPLAR SYSTEMS SELECTION. High-level architectural views of exemplar systems are sources for structural or operational domain relations. In addition, documented counter-exemplars are sources for conceptual domain relations.
- DOMAIN STAKEHOLDER KNOWLEDGE. Elicited informally in order to identify domain systems and development work flow contexts.
- DOMAINS OF INTEREST. A source for possible related domains to be documented in the DOMAIN INTERCONNECTION MODEL. Also, the DOMAINS OF INTEREST LEXICON may be a source of terms for related domains documented in the DOMAIN INTERCONNECTION MODEL.

  Note that DOMAINS OF INTEREST were previously used to determine the DOMAIN SELECTION in the previous sub-phase, *Scope Domain*. In this task we start again from the original list of DOMAINS OF INTEREST to discover related domains.

## Controls

- DOMAIN SELECTION. To the extent that the domain selection task included technology forecasting (i.e., identifying the "technology window" for the domain), this information is used in setting the historical context for the domain.

- PROJECT OBJECTIVES. Even though identification of related domains and contexts for the domain of focus is one strategy to help scope Modeling, this process itself needs to be filtered in terms of project objectives. Related domains identified in the DOMAIN INTERCONNECTION MODEL, where possible, should be of strategic significance within the organization context.
- DOMAIN STAKEHOLDER MODEL. Used here as a filter on the application contexts considered.

## Activities

The following activities is to use a variety of complementary techniques to elicit conceptual, structural and historical linkages between the domain of focus and related domains. Of particular interest are "neighboring" domains, in terms of conceptual closeness, structural similarity or close integration, and historical lineage. They are not intended to be performed in strict sequence or necessarily to the same level of detail for every domain.

Conceptual or semantic relationships between domains can be determined by at least three relevant types of information:

- Exemplar Systems. Each domain can be associated with a set of exemplar systems within the broader set of systems of interest. By noting set—subset relations in the exemplar sets associated with different domains, semantic relationships between these domains can be established. For example, for the two domains "Interactive Outline Browsers" and "Interactive Outline Browser-Editors", the second would map to a subset of those systems mapped to the first domain. Some outline browsers have editing capabilities, while some (e.g., outline-oriented directory navigation interfaces) do not.
- Defining Features. Each domain can be associated with a set of defining features and rules of inclusion and exclusion. Domains that are associated with subsets of the defining features of other domains also have identifiable semantic relations.
- Organization Context. Domains can be associated with the organization contexts in which they are defined. Thus, in contextual terms, the domain of "Army Command and Control Systems" is a narrower domain than "Military Command and Control Systems".

➤ Identify candidate related domains

Identify an initial set of candidate domains that appear significantly related to the domain of focus. Record these interim results in the DOMAIN AFFINITY DIAGRAM. Develop the initial list of candidate related domains from the following sources:

- Filter the DOMAINS OF INTEREST, identifying that subset of domains connected to the domain of focus via the relation types described in the following activity description. Not all domains in this set will have relevant relationships to the domain of focus, nor will all related domains appear in this set. Check the DOMAINS OF INTEREST lexicon for terms suggesting related domains, particularly looking for terms borrowed from analogous domains.
- For each of the systems in the EXEMPLAR SYSTEMS SELECTION, identify principal interfaces to domain functionality in the system. Work from high-level architectural descriptions if they are available. These interfaces should suggest *structurally* related domains. Find a provisional domain name that encompasses the system functionality in question (e.g., window manager, database system) and document it.

- Working specifically with the counter-exemplars documented in the EXEMPLAR SYSTEMS SELECTION, try to find domain names that correlate with the counter-exemplars.

  *Example.* In the Outlining domain, a to-do list manager program may have been considered as a candidate exemplar, then excluded because of the domain defining rule, "Outlining programs can support at least three levels of indentation." Noted as a counter-exemplar, in this activity related domains of "to-do list managers", or "two-level outline browsers", might be identified. Note that these two possibilities

- Working from the DOMAIN STAKEHOLDER MODEL identify application contexts in which domain functionality is developed or used. Elicit related domains to consider from stakeholders, using questions based on the relation types described below.

- Working from the DOMAIN DEFINING FEATURES, try successively "masking" one required feature away to see whether a recognizable and distinct related domain emerges.

➤ Identify conceptual relations

For each domain identified, characterize its defining features in relation to the DOMAIN DEFINING FEATURES from the INTENSIONAL DOMAIN DEFINITION. For domains derived from the DOMAIN DEFINING FEATURES, or abstracted from candidate exemplar systems previously characterized in the *Bound Domain* task (as recorded in the DEFINING FEATURES MAP), this should be a straightforward mapping. For other domains, some additional stakeholder information may be required. Record the results in the RELATED DOMAINS—DEFINING FEATURES MAP. A template of this workproduct is illustrated in Exhibit C-7.

Based on these different ways of assessing domain relationships, the following specific types of conceptual relations between domains can be documented in the DOMAIN INTERCONNECTION MODEL.

- Specialization domains. A domain D1 is said to ***specialize*** a domain D2 if:
  - The exemplar set, or organization context, of D1 is a subset of the exemplar set of D2;
  - The set of defining features for D1 is a *superset* of the defining features for D2 (i.e., a more tightly constrained definition). In the examples above, "Interactive Outline Browser-Editors" specializes "Interactive Outline Browsers"; and "Army Command and Control Systems" specializes "Military Command and Control Systems".

- Generalization domains. This semantic relation is the inverse of the specialization relation.

  *Example.* The outlining domain supports features that are a subset of the hypertext domain. Thus an outliner can be considered a "special case" or restricted hypertext system. Similarly, a hierarchical directory browser uses a subset of outlining features.

- Analogy domains. In a formal sense, domains D1 and D2 are said to be ***analogy domains*** if they have overlapping sets of associated exemplar systems or defining features, with neither a complete subset of the other. Analogy domains worth documenting are those that have some pragmatic significance to domain practitioners. An analogy domain may have influenced terminology, design choices, or documented explanations for system behavior provided to users for the domain of focus.

  *Example.* In the Outlining domain, one analogy domain is revealed in the use of genealogical terms like "parent heading", "child heading", and "sibling heading". (In some versions, "mother/daughter/sister heading" and even operations like "Create aunt" are used!) Here, the domain of human genealogical relations is employed in the system documentation and user interface to provide end-users a familiar, intuitive domain for reference.

➤ Identify structural relations

- Component Domains. These are functional areas within the internal architecture of the domain functional "slice" within exemplar systems. The purpose of documenting these "internal" domain interconnections is to prevent the later *Domain Model* phase from modeling extraneous system details, perhaps necessary to domain functionality but essentially a separate area of concern.

  *Example.* In the Outlining domain, most outlining programs have functions to generate a numbered outline header for the document (e.g., 1, 1.1, 1.2, 1.2.1, etc.). This sub-function has its own fairly complex set of options for different styles of enumeration, overriding of defaults, etc. This could be considered a component domain. An enumerator is not an "outliner in miniature" in the same way that a to-do list manager is an outliner with many restricted functions.

  On the other hand, note that the "outline enumeration" sub-domain might be a component of other applications besides outliners. This is why component sub-domains cannot be treated like components and sub-systems of traditional systems design.

- Operational Contexts. This relation is basically the inverse of the preceding: i.e., look for contexts where the domain of focus is a component domain of another domain.

  *Example.* In the Outlining domain, some outlining programs are stand-alone, while others are embedded in other applications such as text-editors, presentation managers, and spread-sheet applications. These would become application contexts in the DOMAIN INTERCONNECTION MODEL.

  Closely related to a conventional system context diagram or high-level system architecture, this model depicts the domain of focus as a component, with interfaces illustrated to all related subsystems within the exemplar systems for the domain. (This assumes a subsystem-level domain of focus, that is, a horizontal domain relative to the exemplar systems.) These interfaces often raise canonical design issues, e.g., choice of strong vs. weak coupling across an operational domain boundary such as a client-server relationship. Major system variants that appear in the domain model may be characterized in large part by design decisions at these boundaries.

  Unlike a conventional system architectural model, the Operational Context model contains the superset of all major interfaces to domain functions, even where no single exemplar system includes or could include all interfaces simultaneously.

➤ Identify historical and life cycle relations

Document the varied contexts of use for domain functions, working from the DOMAIN STAKEHOLDER MODEL, and eliciting supplementary DOMAIN STAKEHOLDER KNOWLEDGE as required.

- Domain Life Cycle Contexts. The software engineering life cycle specific to the organization and domain of focus determines a set of distinct contexts for development and use of exemplar systems. At a minimum, a given domain of focus will involve three contexts: ***system-in-development***, ***system-in-operation***, and ***system-in-use***. There may be contexts in the life cycle unique to each organization and domain.

Related domains are discovered by examining the processes performed and products generated in each context. For example, most application domains would have a related test data domain; testing will typically be a function in the development and not the operational environment. Documenting potential contexts of use are an important aspect this picture. These contexts will be more thoroughly modeled once a specific set of representative systems are selected in the *Plan Data Acquisition* task.

- Domain History. Document the historical emergence and evolution of the domain as a recognized abstraction in the community of practice. This could involve documentation of predecessor and/or successor domain technologies, new technologies that represent a next step beyond the domain's current technology, etc.

  Analogical and historical relations may overlap closely. Many automated systems retain the older terminology of superseded manual processes. For example, many computer typesetting systems retain terms and operations from hot-metal printing technology, although the terms often imply a physical mechanism and associated restrictions that is no longer relevant.

  (Note that this historical view is distinct from information about the historical relations among exemplar systems in the domain. This information is gathered in the *Plan Data Acquisition* task that begins the next sub-phase, *Acquire Domain Information.*)

➤ Integrate the relational views

As the final step in this task, synthesize the various views of domain relations, looking for connections between structural, conceptual, historical, and life cycle views. One example of how to do this integration is illustrated in the template shown in Exhibit C-8, Conceptual — Historical Relations Map. (This is a suggested and not a required workproduct.)

Domains related conceptually are mapped against a rough time line showing the evolution of the domain. One natural historical trend is for a system to spawn specializations as spin-offs; value-added extensions would be a classic illustration of this. Each extension addresses a *narrower* scope of application, but provides *greater* functionality to support that scope of application. Conversely, a generalization trend reflects developers' discovery that by *removing* certain functions, a system becomes usable in new contexts.

## Workproducts

■ DOMAIN INTERCONNECTION MODEL

The DOMAIN INTERCONNECTION MODEL defines relationships between domains, not between systems or system components. This makes it quite different from a conventional systems model or a model of assets in an asset base.

It is developed using the following interim workproducts:

- DOMAIN AFFINITY DIAGRAM. An informal graphic depiction of the domain of focus in the center of numerous overlapping related domains. It is used as a preliminary way of capturing candidate domains before they are classified more formally.

  In some cases it may be possible to name the intersection of the related domain with the domain of focus, as in the nature of a Venn diagram. This is not required, however. Cross-relationships *between* related domains are not systematically documented in the picture. Although placement and conjunction may be used to suggest intuitive relations, the diagram does not attempt to classify the related domains. In fact, it more closely resembles a conceptual association network than a structural system components model. (Not illustrated.)

THE DOMAIN AFFINITY DIAGRAM includes the RELATED DOMAINS — DEFINING FEATURES MAP. The RELATED DOMAINS — DEFINING FEATURES MAP is used to characterize and classify related domains in terms of the DOMAIN DEFINING FEATURES from the INTENSIONAL DOMAIN DEFINITION. A template of this workproduct is illustrated in Exhibit C-7.

- CONCEPTUAL — HISTORICAL RELATIONS MAP. Classify related domains into specialization domains, generalization domains, and analogy domains. A template of the CONCEPTUAL — HISTORICAL RELATIONS MAP is illustrated in Exhibit C-8.

The final workproduct contains these key elements (these have been described in the Activity sections above):

- Conceptual Relations.
  - Specialization domains
  - Generalization domains
  - Analogy domains
- Operational/Structural Contexts.
  - Component sub-domains
  - Operational Contexts
- Domain Life Cycle Contexts.
  - system-in-development contexts
  - system-in-use contexts
- Domain History.
  - Predecessor systems
  - Forecasts of new systems

## Guidelines

- Use analogy domains as an elicitation technique. Probing for analogy domains in interactions with domain informants can be facilitated by a kind of controlled brainstorming that pushes analogies in surprising ways. The process can help to unearth what may have been unconscious design choices and rationale, along with occasional oversights. When an analogy relation is discovered, consider these questions:
  - What features of the analogy domain have been carried over by designers, though not needed in this domain?
  - What features were not transferred, but would be relevant in this domain?
  - What features relevant to this domain have been inadequately addressed because they do not fit the design analogy used?

This technique can be used particularly effectively by domain analysts who are *not* experts in the selected domain (or who are experts at hiding their expertise).

- Since PROJECT OBJECTIVES and available resources will rarely allow consideration of the entire set of contexts relevant to a given domain of focus, this modeling step will often give rise to new boundary issues that will result in iterative refinement of the INTENSIONAL DOMAIN DEFINITION and EXEMPLAR SYSTEMS SELECTION.

## Exit Criteria

- The *Orient Domain* task is the final task in the *Plan Domain Engineering* phase as a whole. As such, it serves as a kind of summary for much of the information gathered throughout the phase, and its completion criteria indicate some of the completion criteria for the phase as a whole.
- Assess the domain boundaries that result from *Define Domain* sub-phase with respect to PROJECT CONSTRAINTS (e.g., resources, schedule, staffing, and budget). The relation suggests whether it is appropriate to proceed on to the *Model Domain* phase. The following scenarios indicate the nature of the choices:
  - Domain a little too large. You can probably proceed to descriptive modeling. The initial planning step for descriptive modeling will be able to tune or trim down the model coverage in a variety of ways.
  - Domain a little too small. Probably the ideal scenario, since descriptive modeling will invariably unearth more detail than you expect. Proceed to detailed modeling. It will always be possible to iterate to improve quality or depth of detail in the model.
  - Domain much too large. Iterate the *Plan Domain Engineering* phase to define a narrower focus within the selected domain.

    In order to re-enter the planning loop, the domain-of-focus centered view must be converted to the "ORGANIZATION CONTEXT" view required as an input to the planning tasks. However, this context will now probably be documented much more explicitly than on the first iteration.
  - Domain much too small. The project's process may be broken; i.e., you may have iterated too far. Consider backing out to a wider focus; this will be easier if sufficient traceability and rationale was retained in workproducts to this point.

    It is also possible that the process was basically sound, and that the natural domain boundaries fall awkwardly (i.e., last iteration produced a domain of focus too large, but sub-domains are all too small). In this case, several strategies can be applied, including:
    - Aggregating several related domains into one descriptive modeling effort (such domains should probably share a primary operational interface, so that the results can be integrated to form a larger functional piece, but also separately reused);
    - Changing the project plan to iterate through a series of smaller domain modeling efforts. This approach makes particular sense if learning about and refining the domain engineering process is an explicit objective.
    - Spawning several smaller-scale parallel domain modeling efforts within the overall project scope. Select a parallel domain by reentering the *Plan Domain Engineering* phase at the *Select Domain of Focus* task, reusing the DOMAINS OF INTEREST report

and other workproducts from earlier steps. In parallel proceed with descriptive modeling for this domain.

## Validation and Verification

- Qualify borderline and counter exemplar systems. Just as intensional and extensional definitions served as cross-validation during *Bound Domain*, the DOMAIN INTERCONNECTION MODEL can be used to cross-validate DOMAIN EXEMPLAR SYSTEMS. The set of borderline and counter exemplar systems identified are not further characterized in relation to the domain of focus within *Bound Domain*. Within *Orient Domain*, however, borderline and counter exemplar systems can be characterized in terms of related domains. Where no related domain can be associated with an exemplar system, the set of related domains is examined for completeness and consistency; new related domains may be added. In addition, related domains with no borderline or counter-exemplars may result in additional investigation of candidate exemplar systems.

  If traceability is being performed, the results of the validation can be documented in annotations to the EXEMPLAR SYSTEMS SELECTION.

- Document domain boundary decisions in an optional DOMAIN BOUNDARY DECISION REPORT. As decisions about domain boundaries shift in the course of the project, this report can capture process history, decisions and rationale for the shifts. This information is also valuable for future lessons learned. and for validation and modification of related workproducts to assure consistency of all ODM workproducts during modeling changes.

# 6.0 Model Domain

## Purpose

The primary purpose of the *Model Domain* phase of the ODM domain engineering life cycle is to produce a DOMAIN MODEL for the selected domain, based on the DOMAIN DEFINITION produced in the *Plan Domain Engineering* phase. The key role of the DOMAIN MODEL is to describe common and variant features of systems within the domain and rationale for the variations. The DOMAIN MODEL is subsequently used in the *Engineer Asset Base* phase as a basis for selecting the range of variability to be supported by assets in the ASSET BASE.

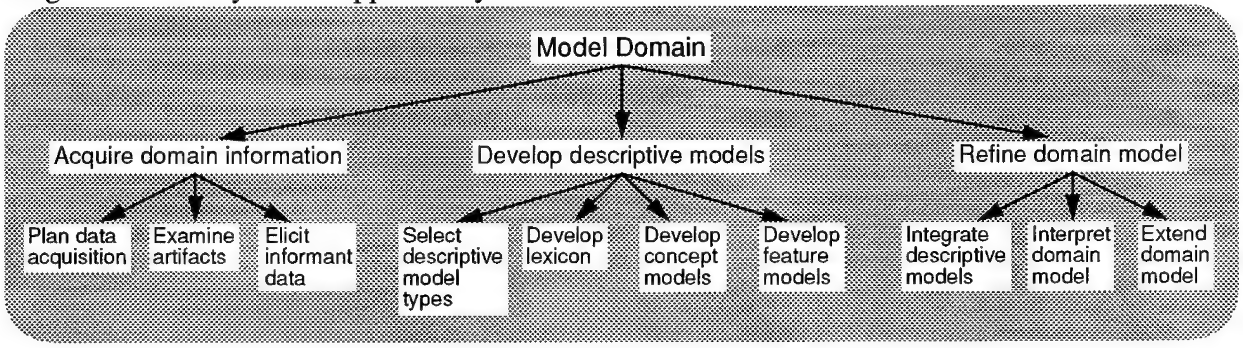


**Exhibit 27.** Model Domain Process Tree

Besides its use in the *Engineer Asset Base* phase, the DOMAIN MODEL can be used in a number of engineering activities outside the domain engineering life cycle described here. This includes use of the DOMAIN MODEL as:

- A resource for training new personnel in the domain and retaining expert knowledge; and
- A basis for directly engineering systems for single application contexts, using a ***model-based development*** approach.

A secondary product of this phase is the DOMAIN DOSSIER which documents the specific information sources used as a basis for modeling. The DOMAIN DOSSIER is be used during the *Engineer Asset Base* phase to trace legacy artifacts that are candidates for reengineering into reusable assets and to identify constraints in systems that assets will be migrated into.

## Description

As shown in Exhibit 28, the *Model Domain* phase consists of three main sub-phases:

- The *Acquire Domain Information* sub-phase involves filtering relevant domain information derived from system ***artifacts*** or from interviews with domain ***informants***. Acquired information is retained in the DOMAIN DOSSIER. The terminology used by domain practitioners is documented in the DOMAIN LEXICON.
- The *Develop Descriptive Models* sub-phase involves the core modeling activities that transform domain data into models of ***commonality*** and ***variability***. In practice, a number of separate DESCRIPTIVE MODELS are developed that capture different aspects of the domain.
- The *Refine Domain Model* sub-phase involves integrating separate DESCRIPTIVE MODELS into a single coordinated DOMAIN MODEL. This model includes interpretive data about the rationale for the commonality and variability observed in systems in the domain. In addition,

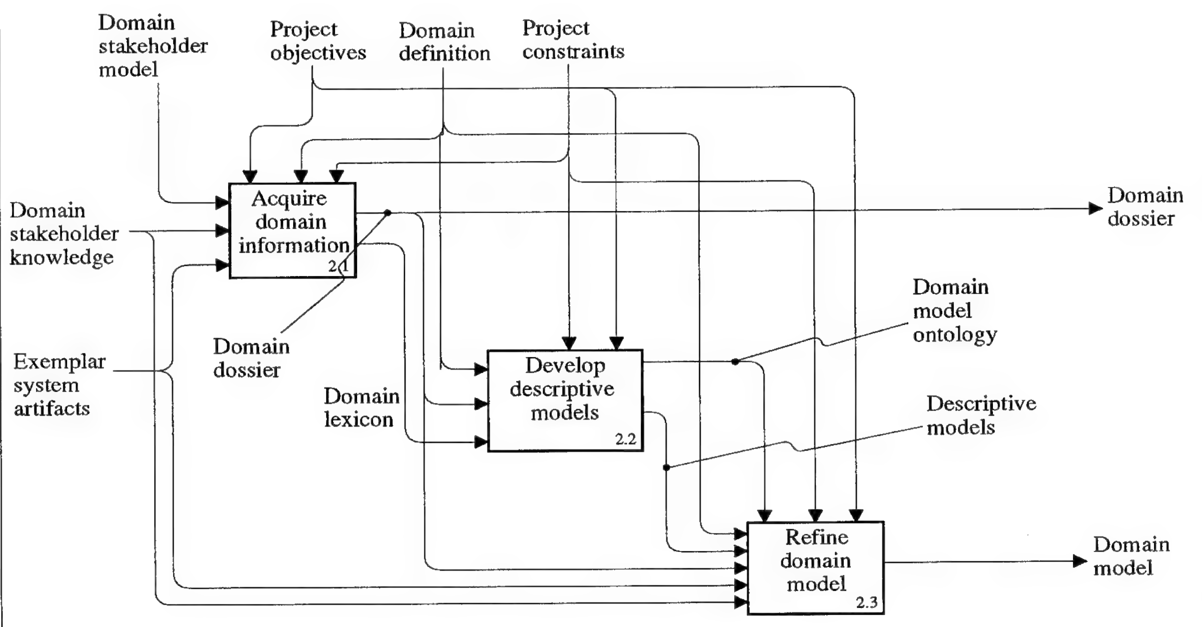


**Exhibit 28.** Model Domain IDEF$_0$ Diagram

the model can include ***innovative features*** suggested during previous modeling activities.

By structuring this phase in terms of these distinct sub-phases modelers can better distinguish subtle transitions in the modeling process that are often a source of errors. When data acquisition and modeling are freely intermixed there are dual risks of either of generating too much data without a comparative framework or jumping into modeling so quickly that insufficient data is considered. When the purely descriptive modeling activity and the refinement activities of integration, interpretation, and innovation are interwoven without distinction, it becomes difficult to assess which aspects of the models reflect design decisions imposed, sometimes unwittingly, by domain modelers. Since modelers are often not the primary practitioners in the domain, they can easily produce models that do not reflect the language or rationale of the domain's ***community of practice***.

*Key Concepts of Domain Modeling*

Before reading the detailed sub-phase and task descriptions for this phase, it is helpful to review a few general points about the nature of the DOMAIN MODEL:

- The DOMAIN MODEL includes commonality and variability. Some approaches to domain engineering assume a DOMAIN MODEL includes *only* common features within the domain. In ODM the DOMAIN MODEL includes *both* common and variant features of the domain. To set boundaries for how much variability to incorporate, modeling is based on information acquired about a specified set of exemplar systems called the REPRESENTATIVE SYSTEMS SELECTION. The range of variability described in the DOMAIN MODEL is validated with respect to this explicit representative set.

- The DOMAIN MODEL can include design and implementation features. Some approaches to domain modeling assume that a DOMAIN MODEL includes only high-level system requirements or functional requirements. While the DOMAIN MODEL in ODM does as a rule include such requirements, it can also include low-level requirements, performance requirements, and design and implementation features. In principle, the DOMAIN MODEL describes the commonality and range of variability for any process or workproduct in any phase of the domain-specific system life cycle.

The relationships and layerings among the descriptive models developed during *Model Domain* are illustrated in Exhibit 29.

## Sequencing

There are many possible ways of sequencing the performance of activities in this phase. Each primary information source undergoes a process of analysis and interpretation and may contribute to multiple, closely interrelated models. Conversely, each model will integrate data observed from multiple information sources. The transformation of information can be performed in broad stages as suggested by the sub-phases, or more incrementally and at a finer level of granularity. The modeling process as a whole can be performed in a number of different sequences:

- The ***ontology-driven modeling*** approach is focused on acquiring data to populate a specific model. In this approach, the modeler starts with a selected model, then sifts through domain information sources for data relevant to this model. One advantage of this approach is that it is easier to control acquiring necessary and sufficient data for each model. One disadvantage is that the same information source may be analyzed numerous times, by different modelers, in the context of different models.

- The second approach is ***information-driven modeling***. In this approach, the modeler begins with an information source, and attempts to filter relevant data from that information source into the appropriate models. One advantage of this approach is that information sources can be very thoroughly modeled, and may suggest new and important model types not originally planned for in the DOMAIN MODEL ONTOLOGY. One disadvantage is the difficulty in regulating the appropriate level of detail to model.

- Various hybrid strategies are also possible. One possibility would involve a matrixed team structure. Each modeler is given responsibility for a set of information sources (perhaps including particular informants as well as system artifacts and documents for particular representative systems). Simultaneously, each modeler is given ownership of a particular model or set of models. Each modeler distributes data acquired from their information sources to other modelers, and receives data in turn for their own models. This approach involves each modeler in both data acquisition and modeling activities. However, the approach requires a high degree of team coordination and ***team modeling*** skills. (See Section 9.4 for a discussion of team modeling techniques as they relate to ODM.)

The sequence chosen will have a dramatic impact on the structure and content of the resulting DOMAIN MODEL. Cognitive processes like memory, analogy, and classification are intrinsic elements in modeling. Modelers' insights into the significant commonality and variability implicit in domain information will be influenced by their relative experience in the domain, the number of contrasting examples seen, the order in which they are examined, and the delay between examining data and developing models.

Rationale about specific alternatives and trade-offs within individual systems in the domain is modeled during the *Develop Descriptive Models* sub-phase. This is one reason why the is a critical intermediary step between *Acquire Domain Information* and *Refine Domain Model*. During

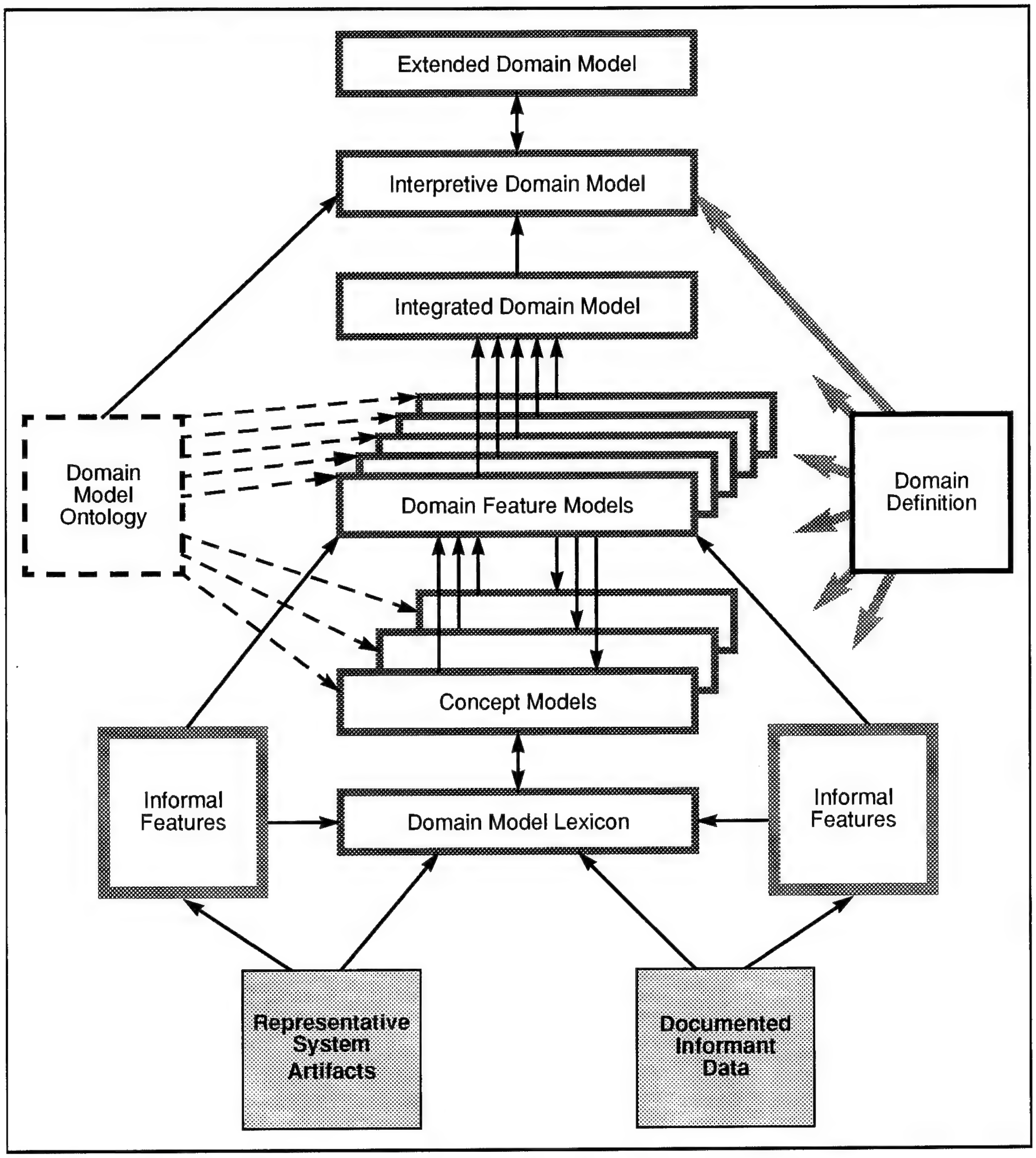


**Exhibit 29.** ODM Descriptive Model Layers

the *Refine Domain Model* sub-phase, rationale is derived for the variability across multiple systems in the domain. This leads to a greater understanding of the domain that can only be gained by looking across systems.

## 6.1 Acquire Domain Information

### Purpose

The primary purpose of the *Acquire Domain Information* sub-phase is to systematically gather information on which to base the DOMAIN MODEL. While some data has been gathered throughout the *Plan Domain Engineering* phase, this sub-phase marks a transition to systematically tracking the derivation of data used as a basis for the DOMAIN MODEL. The *Acquire Domain Information* sub-phase also represents an attempt to comprehensively consider a rich variety of potential sources of information about the domain. Systematic comparative analysis is performed in the subsequent *Develop Descriptive Models* sub-phase. Although domain modelers will learn a

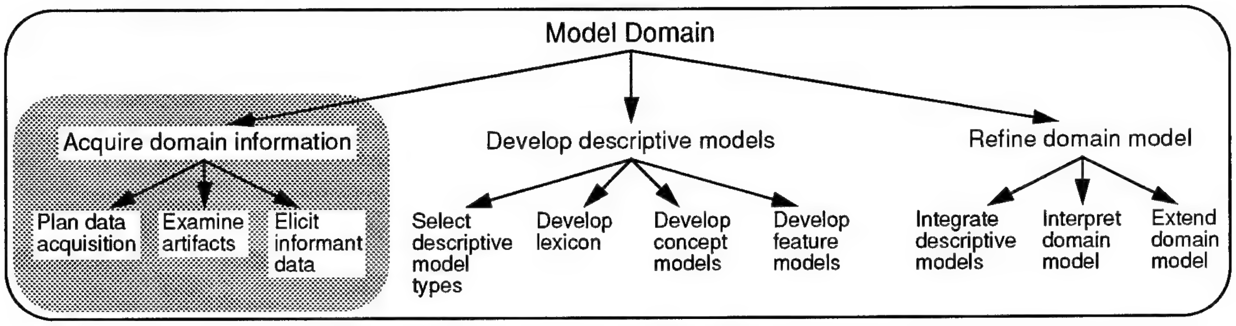


**Exhibit 30.** Acquire Domain Information Process Tree

lot about the domain in this sub-phase, the primary purpose of this sub-phase is *not* merely to increase modelers' knowledge of the domain of focus.

### Description

As shown in Exhibit 31, there are three primary tasks in this sub-phase:

- The *Plan Data Acquisition* task involves scoping the data to be examined in modeling. In this task the REPRESENTATIVE SYSTEMS SELECTION, a subset of exemplar systems for the domain, is selected for detailed data acquisition and modeling. In addition, specific types of data to be acquired and methods of data acquisition are selected. These choices and the specific strategy for acquiring information are documented in the DATA ACQUISITION PLAN.
- The *Examine Artifacts* task focuses on acquiring data through examination of various representative system artifacts from across the system life cycle. An ***artifact*** is broadly defined as a system workproduct selected as a source of information for domain engineering. In most cases using a workproduct as an artifact means interpreting it in ways quite different than its original purpose. For example, a requirements document, when used in the context of developing the system to which the requirements refer, provides a set of specifications that determine the design task. Requirements documents viewed as artifacts by a domain engineer are treated as a source of descriptive data about the range of requirements within different exemplars in the domain.
- The *Elicit Informant Data* task involves gathering information from people in various ways: interviewing, observation, participation in work practice, etc.

The *Examine Artifacts* and *Elicit Informant Data* tasks are broken out as separate processes to emphasize the distinctive activities and skills required for these techniques of data acquisition.

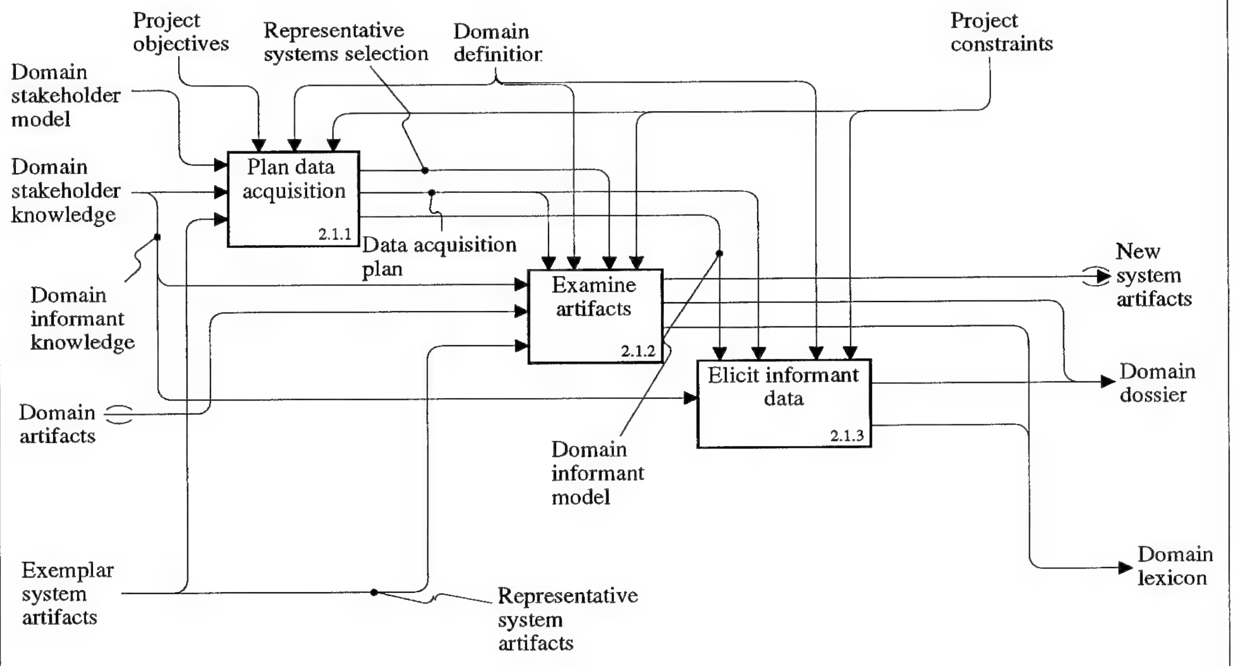


**Exhibit 31.** Acquire Domain Information $IDEF_0$ Diagram

*Modeling System Context*

A key aspect of acquiring domain information is selection of the ***system contexts*** to study for each representative system. Exhibit 32 shows the system contexts in a minimal domain-specific system life cycle. Exhibit 33 shows the contextual layers within a domain. The following paragraphs will provide some background on why we consider these different contexts in the ODM approach to data acquisition for domain modeling.

Like other products and information, software systems are created in, exchanged among and used in organization contexts. What makes the analysis of software systems problematic in this regard is the following:

- Systems span organization contexts. It is inherent in the nature of software that creation and use will take place in different settings. There can be many other settings involved as well (maintenance, tech support, customization, etc.) but the essence of the problem can be seen with the simple distinction between the developer and user environments. Exhibit 32 depicts a minimal schema for a system life cycle that can be tailored to the specific contexts in each domain. Each context can be thought of as the intersection or "hand-off" of system artifacts with people in different organizational settings.
- Systems embed implicit contextual knowledge. The handoff of the system across different organization settings can become a magnet for many process and communication break-downs, because so much contextual knowledge needed for organizations to run effectively becomes hidden to practitioners. When assumptions based on this contextual knowledge are

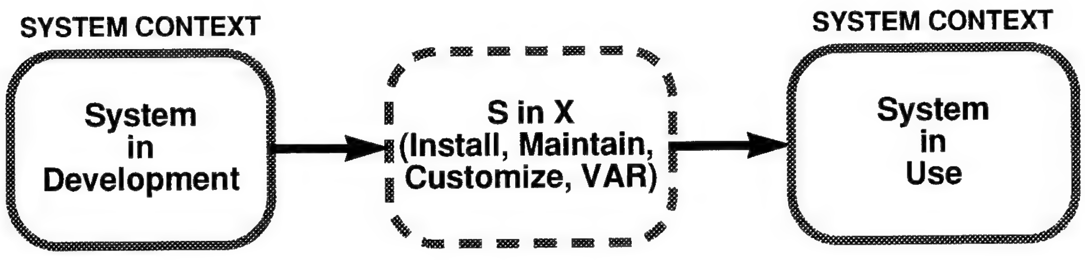


**Exhibit 32.** System Contexts in a Minimal Domain-Specific System Life Cycle

many problems in predictability, quality and reliability result.

While increased awareness of these contextual issues could improve the quality of general software engineering practice, it is an *essential* component of a viable approach to domain engineering. The concept of a software domain introduces an entirely new level of cross-contextual complexity to the engineering problem. Domains are typically defined in terms of multiple sets of software systems. Not only are multiple organization contexts involved in the life cycle and evolution of each individual system, but different systems in the domain will span an even wider range of contexts. (This is particularly but not exclusively true for domains that include systems developed independently by different organizations.)

Thus, domains are shaped by multiple overlapping contexts of development, maintenance, customization, and application. The language, values, assumptions and history of each relevant community of practice introduce contextual information into software artifacts that invisibly constrain those artifacts. The context of artifact development and use in social terms -- usage in work process flow, values about types of languages, use of development tools and development process models, and work environment structure and systems (or at least the developers' views of these) directly influence features of the software. Exhibit 33 depicts some of the embedded contextual layers that surround a piece of software in its operational environment. Anyone who has ever cursed at an inscrutable error message has encountered contextual blind spots in action.

Thus, a primary purpose of the *Acquire Domain Information* sub-phase is to ***re-contextualize*** sys-

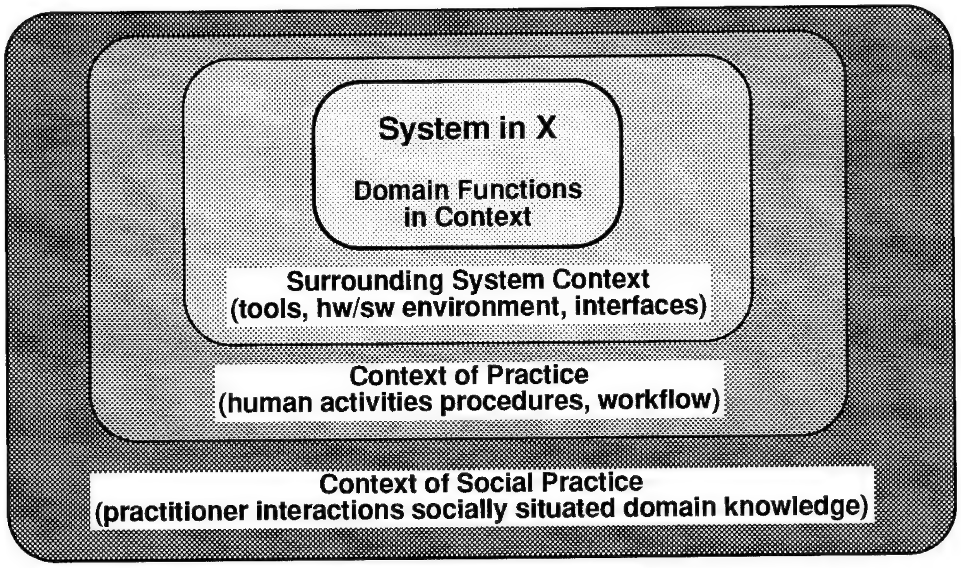

System in X
Domain Functions in Context
Surrounding System Context (tools, hw/sw environment, interfaces)
Context of Practice (human activities procedures, workflow)
Context of Social Practice (practitioner interactions socially situated domain knowledge)

**Exhibit 33.** Contextual Layers Operative in Each Domain Context

tem artifacts: to identify and make explicit the contextual information embedded within artifacts. Embedded assumptions can be found via identifying, describing, and modeling the cultural or contextual aspects of the domain as key descriptors along with software artifacts.

This can render these artifacts more dependable and predictable for *ad hoc* reuse. However, artifacts may not necessarily be suitable for a wide variety of application contexts. They may be engineered with many context dependencies that work against transfer to other contexts. Recontextualizing makes these dependencies explicit, but does not remove them. This requires the further step of reengineering the artifacts into assets, which is the task of the *Asset Base Engineering* phase of ODM.

## Sequencing

- Iteration between planning and data acquisition. Some iteration between *Plan Data Acquisition* and the *Examine Artifacts* and *Elicit Informant Data* tasks will be natural. Each data gathering pass will yield some newly discovered information sources. Informants will suggest other people to talk to. Artifacts will reference other documents. There is a trade-off as to how many newly-emerging data acquisition possibilities can be explored. One extreme approach is to stick closely to the original planned inputs. The other is to be too reactive to new sources of information. A key design task in this process is balancing the need for completeness in data acquisition with the inefficiency of examining redundant data.

  A balance must be maintained that keeps project resources allocated appropriately, but allows for flexibility in planning. This flexibility is especially important when extending the scope will help alleviate the significant risk of skewed data, or will provied significant opportunity for unanticipated extra benefit. One key to managing this balance is to pass new information sources through the filter of the *Plan Data Acquisition* task. If necessary, reprioritize and drop some originally planned data sources if better sources are discovered later, to avoid the expanding resources problem.

  Even given infinite resources, thorough study of all representative systems might not necessarily be useful. Remember that the point is not to study systems for their own sake, but to generate data useful in modeling commonality and variability. Pay close attention for signs that the point of diminishing returns has been reached (e.g., people get bored with acquiring data).

- Representative system data acquisition. A primary choice in sequencing within the *Acquire Domain Information* sub-phase is whether to collect all data for each representative system in one pass, moving on then to the next representative (the representative-by-representative approach); or to collect analogous types of data (e.g., design documents, performance requirements) from across all representatives in one pass (the information type-oriented approach).

  — The advantage of the representative-by-representative approach is that modelers' understanding of the system has a chance to deepen. For complex systems, it takes time to understand the system and how it works. The risk in this approach is of lapsing into *modeling the system*, rather than modeling the system's comparative features with respect to the domain as a whole. This risk is present in all activities of this sub-phase.

  — The information type-oriented approach has the advantage of highlighting comparative data across systems, and encouraging an appropriate level of modeling. One risk in this approach is that modelers' understanding of individual systems may stay too shallow. A particular danger is that this lack of depth in understanding may not become apparent until the *Refine Domain Model* sub-phase, when interpretation of the data is required.

- Acquiring data from artifacts and informants. There are trade-offs involved in determining whether to begin data acquisition with informant interviews or artifact analysis. Interviews with the right informants can help to determine the best artifacts to study. On the other hand, familiarity with exemplar applications through study of artifacts may be necessary in order to create the right set of questions for effective interviewing.

## 6.1.1 Plan Data Acquisition

### Purpose

The primary purpose of the *Plan Data Acquisition* task is to scope the data acquisition effort for the Domain Modeling phase of the domain engineering life cycle. In the final task of the Planning phase, *Define Domain*, domain boundaries were drawn in terms of intrinsic structural and conceptual principles. Based on the same DOMAIN DEFINITION, many different sets of data can be selected as a basis for modeling. The data selected is a specific limited "window" into the knowledge of the domain.

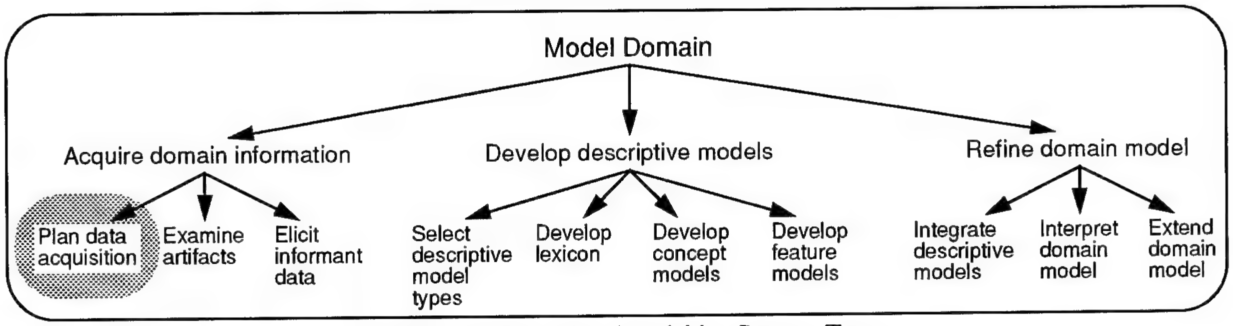


**Exhibit 34.** Plan Data Acquisition Process Tree

In *Plan Data Acquisition*, modelers determine the specific subset of EXEMPLAR SYSTEMS SELECTION to study as ***representative systems***, and select the ***system contexts*** and types of ***information sources*** to study for each representative system. Both the system development and system use contexts are modeled to ensure that the DOMAIN MODEL includes situations where domain functionality can be deployed. This modeling is performed based on the notion of a domain ***feature*** as a differentiating behavior or characteristic associated with a system artifact that is significant to some domain practitioner within a given context. For example, whether a given function requires specific code, a site-determined parameter, a start-up value for an interactive session, or a dynamically specifiable default, will have significant impact on the usefulness of that function to developers and users of the system. The DATA ACQUISITION PLAN ensures a degree of balance and comprehensiveness in data acquisition.

In the overall context of the domain engineering life cycle, the *Plan Data Acquisition* task provides:

- documentation of the empirical data that forms a basis for evaluating the completeness and consistency of the DOMAIN MODEL to be produced; and
- a strategic framework for what data will be gathered about the domain, based on PROJECT CONSTRAINTS and characteristics of available domain information sources.

## Entrance Criteria

- Domain of focus has been selected. If this task begins before domain of focus selection the risk is that excessive resources will be applied in characterizing information availability for all systems within the organization context (i.e., more than just exemplar systems for the selected domain). Such an inventory might have value to the organization but goes beyond the needs of domain engineering.
- Domain of focus has been bounded. The *Bound Domain* task has produced an initial INTENSIONAL DOMAIN DEFINITION and EXEMPLAR SYSTEMS SELECTION and they have been cross-validated for consistency. If the process begins before the domain boundary is reasonably stable, there will be insufficient criteria for selecting representative systems or relevant information types.

Note that the DOMAIN INTERCONNECTION MODEL need not be complete before this task begins.

## Inputs

- DOMAIN STAKEHOLDER MODEL. Each category of stakeholders is assessed in terms of:
  - the value of the stakeholders as informants; and
  - what can they tell us about the domain.

  For example, distinctions could be made between novice, journeyman and expert experience levels in the domain.

  THE DOMAIN STAKEHOLDER MODEL, refined with this assessment information, becomes the basis for the DOMAIN INFORMANT MODEL.
- EXEMPLAR SYSTEM ARTIFACTS. Artifacts available for each exemplar system. These artifacts are inventoried and classified by type of artifact.
- DOMAIN STAKEHOLDER KNOWLEDGE. Other artifacts and informants available for the domain is inventoried. These include secondary data sources (survey articles, textbooks, etc.) that are not be associated with particular exemplar systems.

## Controls

- PROJECT OBJECTIVES. Both overall objectives and domain-dependent objectives refined during the *Select Domain of Focus* task provide a basis for filtering and prioritizing the data to be acquired.
- PROJECT CONSTRAINTS. PROJECT RESOURCES will assist in determining how many representative systems can feasibly be studied and what artifacts will be feasible to examine given the experience and skill level of the domain engineering team staff.

  If PROJECT CONSTRAINTS include specific staffing and scheduling information, this information may flow into the DATA ACQUISITION PLAN in the form of specific team assignments for interviewing and artifact analysis.
- DOMAIN DEFINITION. The EXEMPLAR SYSTEMS SELECTION component, which documents all systems considered during *Define Domain*, is the source for the REPRESENTATIVE SYSTEMS SELECTION.

## Activities

The Plan Data Acquisition task involves scoping domain data to be acquired. This scoping is accomplished in three levels:

- the set of REPRESENTATIVE SYSTEMS to be studied;
- the specific system contexts to investigate for each representative system; and
- the types of information to study for each representative system context.

  *Example.* Out of a set of twenty exemplar systems in the domain, seven representative systems are selected for detailed study. Of this set of seven, requirements and design phase artifacts will be studied for all systems. Three systems will also be investigated in terms of end-user trouble reports and enhancement requests. One of these systems will be studied exhaustively. The other two will be sampled selectively.

The following activities detail the process for producing the DATA ACQUISITION PLAN. These activities are not strictly sequential. In particular, there will be a lot of interaction between characterizing various candidate systems and types of information to be studied.

Detailed project planning for data acquisition is complex and different in nature from planning for typical software development activities. Most of the activities below will be best performed using guidelines drawn from the Information Acquisition Techniques supporting method. This supporting method is described in Section 8.2.

➤ Select Candidate Representatives

To control the effort expended in the *Plan Data Acquisition* task, filter the EXEMPLAR SYSTEMS SELECTION by identifying viable candidate systems for detailed study. Document the viable candidates in the CANDIDATE REPRESENTATIVE SYSTEMS list.

➤ Develop individual system selection criteria

Derive criteria for selecting representative systems from PROJECT OBJECTIVES, including the DOMAIN SPECIFIC PROJECT OBJECTIVES. Document the results in the list of REPRESENTATIVE SELECTION CRITERIA.

- Potential factors to consider when selecting individual representative systems include:
  - — *Data Coverage.* Relative availability of data for each candidate representative system. Are there enough different information types to research for a given candidate representative system?

    *Example.* A legacy system has code but no formal design documents available. PROJECT OBJECTIVES include building a design decision model that will require extensive modeling of design rationale.
  - — *Position of the exemplar system within the EXEMPLAR SYSTEMS SELECTION.* Is the system core or close to the borderline of the domain boundary?
  - — *Data Accessibility.* How easy will it be to access to information for this system?
  - — *Data Quality.* What is the overall quality of the system as an example to study?

- Potential factors to consider for the set of representative systems as a whole include:
  - *Stretch.* Inclusion of corner cases and a wide range of diversity. One type of diversity is structural diversity (e.g., domain functionality includes both encapsulated and distributed implementations).
  - *Finesse.* Select at least a few systems that are functionally very similar. This will ensure that the DOMAIN MODEL will be rich enough to express subtle variations in domain functionality as well as coarse-grained variation.
  - *Comparability.* For a given information type, is there a rich enough cross-section of data available from the different representatives selected? For example, can we study requirements documents from at least three systems?
  - *Completeness of coverage.* Does the set of candidates as a whole provide adequate coverage across the desired range of data?

➤ Map project and candidate time lines

Information that can be acquired for a given representative depends heavily on the current phase of the system's life cycle. Draw a time line that maps the various life cycle phases and project schedules to time for each candidate representative. If all the systems have been fielded, the approximate vintage of the systems should still be indicated. Map this comparative time line against the expected schedule for the domain engineering project. This will yield a means of factoring schedule-driven opportunities and constraints into the representative selection process. Document the results in the PROJECT - REPRESENTATIVES TIME LINE. A template of this workproduct is illustrated in Exhibit C-9.

The set of representative systems should span a reasonable historical range, including:

- existing systems (where source workproducts are available for analysis),
- systems currently under development,
- known requirements for new or potential systems, and
- forecasts of anticipated new areas of demand in the application domain, derived from marketing studies and analysis, strategic planning, etc.

Deciding to include an older system as a representative does not constitute commitment to adopt that system's architecture or design, or to design for reuse in a manner allowing for incorporation of assets into that system. Reengineering may not be possible or even desirable for some legacy systems, yet they may still be worth some degree of modeling. Many system users have been annoyed by cases where newer versions of systems abandoned valuable features of older versions.

Representative systems need not be limited to legacy systems. They will serve as empirical data points in determining the range of variability in the domain, not just as sources for legacy artifacts to reengineer. The legacy systems to reengineer will be determined later, in the *Engineer Asset Base* phase.

➤ Document Genealogy of Candidates

Document the historical interrelationships and dependencies among candidate representative systems. Documenting these historical relationships, at least at a large-grained level, helps ensure selection of a representative set with sufficient diversity of dependent vs. independent development.

Useful genealogical relations include the following:

- *direct successor.* System A superseded system B, adding new features, correcting errors, and observing constraints on compatibility.
- *leveraged.* Artifact (code, and/or design, tests, etc.) from system A was adapted for use in system B, possibly to implement the system for a new customer.
- *requirements reuse.* Operational features of system A were adopted for system B (typical for software products of competitors who begin "feature wars").
- *independently developed.* Systems were implemented in different contexts, and without reference to each other (e.g., applications developed within separate organizations).
- *competitively developed.* Systems were developed for the same target environment (e.g., shadow projects, redundant systems development for reliability, competitive designs to published requirements, applications developed in educational settings).

The set of representatives should include as wide a cross-section of these relations as possible, with particular emphasis placed on independently developed systems. Commonality observed across such systems is likely to represent core functionality for the domain. Avoid selecting a set of representatives that are all members from one systems family. However, intentionally selecting two systems of close lineage as part of the representatives may yield valuable comparative data.

Later in the modeling process (particularly during the *Interpret Domain Model* process) this historical information will help modelers to interpret common and variant features that were observed across representative systems.

➤ Characterize candidates

Qualify the candidate representatives with respect to the criteria identified. Document the results in the SELECTION MATRIX. A template for this workproduct is illustrated in Exhibit C-10.

➤ Select representatives

After characterizing candidate representatives in terms of information sources available, genealogical relations and any other relevant criteria, prioritize candidate representatives. Factor in PROJECT CONSTRAINTS to make the final REPRESENTATIVE SYSTEMS SELECTION. Document selection rationale that might be helpful if later iterations of domain modeling expand or modify the REPRESENTATIVE SYSTEMS SELECTION. For example, if this representative system is dropped from the data acquisition schedule, what functional role was it playing in the representative set as a whole and how will the loss be compensated?

➤ Select system contexts to study

For each representative system, determine the system contexts to be studied. Record the results in the DOMAIN CONTEXTS MAP. A typical System Contexts Map is shown in Exhibit 35 and an example DOMAIN CONTEXTS MAP is shown in Exhibit 36.

The DOMAIN CONTEXTS MAP identifies various contexts in which domain functions are performed. Each context provides the visibility and interests of various domain practitioners. The

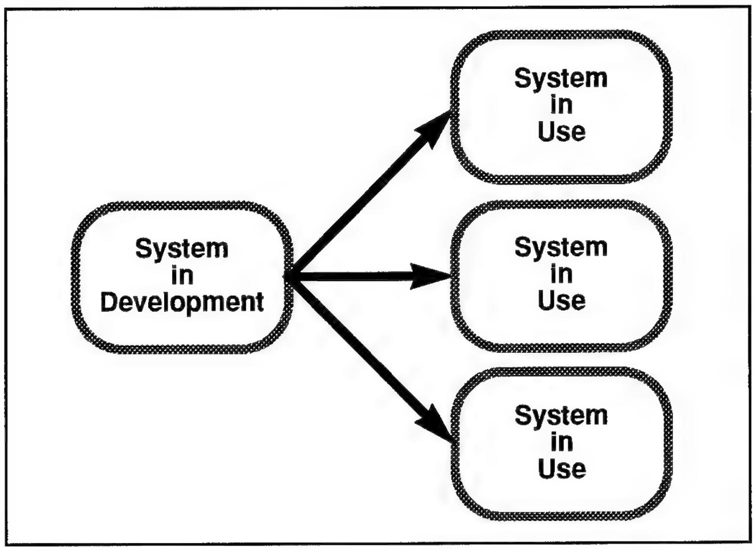


**Exhibit 35.** Typical Exemplar System Contexts Map

DOMAIN CONTEXTS MAP shows the domain-specific set of contexts that characterize the system life cycles. It is refined from the DOMAIN STAKEHOLDERS MODEL and relevant facets of the DOMAIN INTERCONNECTION MODEL. At a minimum, System-in-Development and System-in-Use Contexts should be defined in the model. For most domains, there will be other types of relevant contexts as well.

At a minimum, the following types of system contexts should be considered:

- System-in-use (SIU) contexts. The environments in which end-users execute software systems. Most typically, models of system-in-use contexts describe the way that software-based system components behave in the operational environment for which they are targeted. A host of different notations, methods and representations have been adopted for these models. ODM does not really depend on any one representation being employed to describe the system-in-use context. These are considered part of the System Modeling supporting method area. Supporting methods are discussed in Section 8.

- System-in-development (SID) contexts. The environments in which software developers build systems. System-in-development contexts are typically modeled using some kind of Process Modeling techniques and tools (considered another supporting method area as described in Section 8).

As illustrated in Exhibit 35, even a single exemplar system may have multiple contexts of each of these types. For example, outlining programs may be used by students, collaborating book authors, programmers, etc. In this activity, modelers need to decide which set of these contexts to consider as sources of information.

Domain-Specific Context Types. In addition to these "generic" context types, each domain (and possibly each system) has a unique set of Context Types that refine this simple two-phase model of developer - user into categories that are as domain-specific as possible.

*Example.* The laser printer domain, in addition to the software developers and the end-users, have intermediary roles such as the installer of the font cartridge and an office sys-admin role (who might configure the memory on the printer). These different stakeholders or practitio-

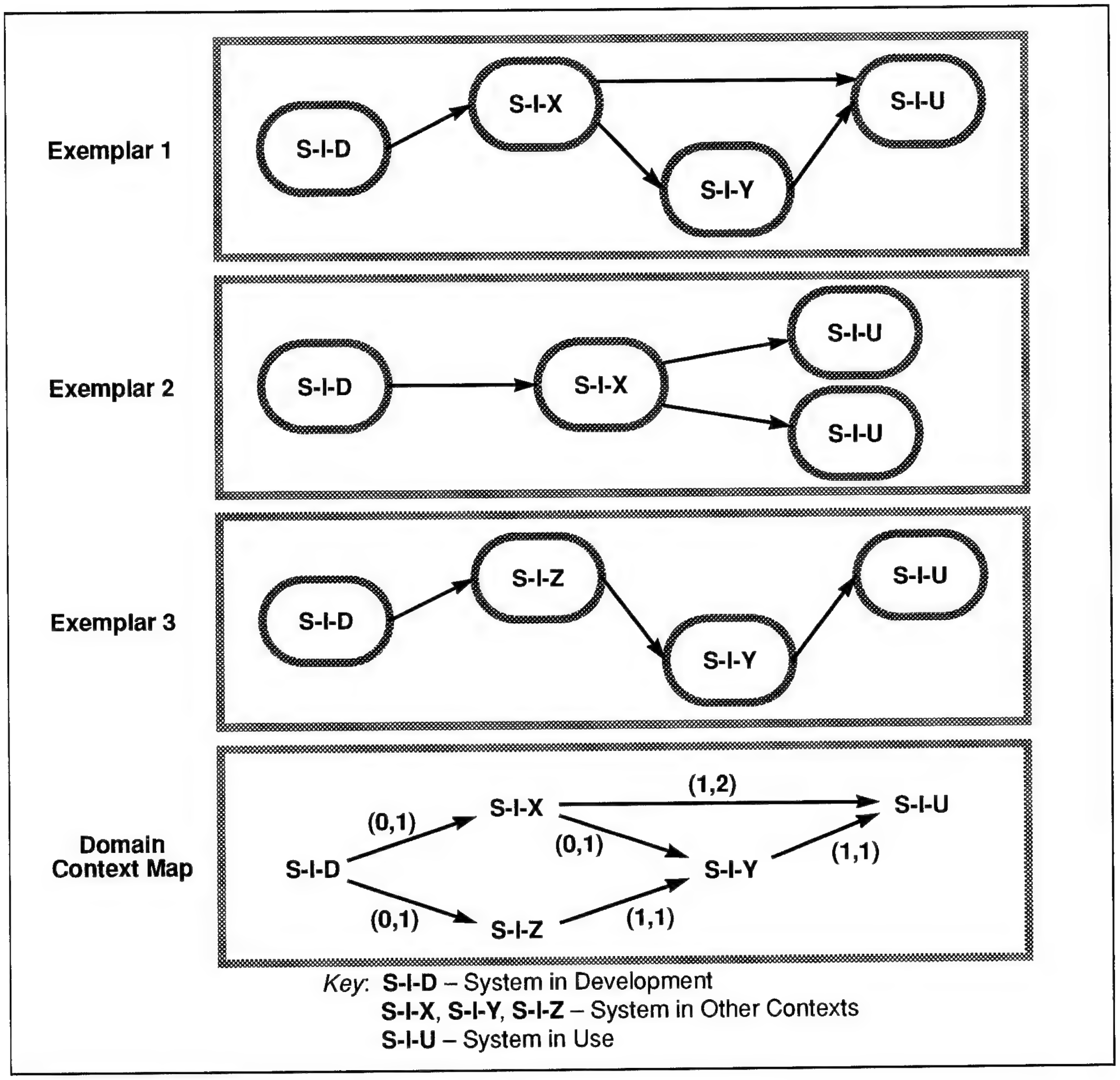


**Exhibit 36.** Example Domain Contexts Map

ners are important to know about; each performs certain tasks that reflect feature "bindings". For example, one could consider the feature of support for a given font in a printer driver. The fonts could be burned into the ROM within the printer, installable via a font cartridge, downloadable in a system configuration mode, downloadable per print job. Or even, if only one character in a font were required for a job, the digital data for this character could be transmitted out of the context of the entire font.

The DOMAIN CONTEXT MAP documents a superset of all the context types occurring at least once in some exemplar system. Preparing this map will require some informal and high-level commonality/variability analysis; e.g., one exemplar's "development environment" may not map directly to a context or environment in another system. The mapping is largely lexical; i.e., what is called the "maintenance" group in one system may be called the "evolution" group in another; what two different systems call "maintenance" groups may need to be distinguished as system contexts. As shown in Exhibit 36, context types and the ways that they can interconnect in exemplar system life cycles can be conveyed in fairly concise notation. (In various templates and figures, S-I-X will denote a domain-specific context type.)

➤ Characterize information types

- Inventory information types available. For each system context in the domain, catalog the:
  - — Artifact types (primary data, workproducts generated and used by domain practitioners)
  - — Secondary data (surveys, historical analyses, industry information, textbooks)
  - — Key processes
  - — Key practitioner roles

  Using the information types inventory, characterize each candidate system in terms of overall availability of information of that type. Code according to accessibility and presumed quality of the artifacts. Record the results in the INFORMATION ACCESSIBILITY — QUALITY MAP that is incorporated into the final DATA ACQUISITION PLAN. A template for this workproduct is illustrated in Exhibit C-11.

- Map informants to representative system contexts. Record the results in the INFORMANT ROLES IN CONTEXT matrix. A template for this workproduct is illustrated in Exhibit C-12.

  ***Domain Informants*** are individuals affiliated with specific stakeholder organizations who are accessed as sources of information for domain engineering. Informants' knowledge and experience can be characterized in terms of roles they have played within various contexts for different systems. For example, a veteran designer could be characterized as having played a role in a number of the design efforts performed by a given company.

➤ Select Information Types

Based on the various workproducts created in earlier activities, make the final selection of information types to be acquired for study. Document the results in the PRIORITIZED INFORMATION TYPES. This can be an annotated refinement of the INFORMATION ACCESSIBILITY —QUALITY MAP and the INFORMANT ROLES IN CONTEXT workproducts where the various rows and columns represent the filtered choices instead of representing candidates.

For each representative system and each system context (i.e., System-in-Development, System-in-Operation, System-in-Use, and other domain-specific contexts), select the types of artifacts to be examined, processes to be documented, and informants to be interviewed. Prioritize the information sources based on PROJECT CONSTRAINTS. Document the results in the PRIORITIZED INFORMATION TYPES matrix.

Some types of information sources that were originally desired may need to be abandoned because of the representatives selected. For example, there may have been too few documented test cases available to do comparative modeling of these. Conversely, the inventory of some candidate representatives might have revealed unique information sources available only for that system. If the PROJECT OBJECTIVES contain mandates that certain types of assets be produced in the *Engineer Asset Base* phase, map the types of information sources that would be of interest for developers of those types of assets.

➤ Structure Data Acquisition Teams and Schedules

Select data acquisition teams, based on PROJECT CONSTRAINTS including PROJECT RESOURCES, and assessed skills of team members. Schedule and prioritize the artifacts and informants to examine. Record the results in the DATA ACQUISITION PLAN.

## Workproducts

■ REPRESENTATIVE SYSTEMS SELECTION

- CANDIDATE REPRESENTATIVE SYSTEMS. The initial filtered list of candidate exemplar systems.
- REPRESENTATIVE SELECTION CRITERIA. Including general and project-specific criteria.
- PROJECT — REPRESENTATIVE TIME LINE. Indicates the schedule relationships between the domain engineering project data acquisition tasks and representative system project schedules. A template of this workproduct is illustrated in Exhibit C-9.
- SELECTION MATRIX. Maps candidate representatives to selection criteria. A template of this workproduct is illustrated in Exhibit C-10.

■ DOMAIN INFORMANT MODEL

The DOMAIN INFORMANT MODEL includes the INFORMANT ROLES IN CONTEXT, a map of who knows what about the domain; i.e., informants mapped to representative systems about which they have knowledge and experience. A template of the INFORMANT ROLES IN CONTEXT is illustrated in Exhibit C-11. Also includes the practitioner roles which potential informants have played in each relevant system context. Informants not associated with specific representative systems (e.g., experts with experience with multiple systems) are ranked in a similar way. The Domain Informant Model is created during the *Plan Data Acquisition* task in the Domain Modeling phase.

■ DATA ACQUISITION PLAN

Plan containing the information sources to be investigated for the domain.

- INFORMATION ACCESSIBILITY — QUALITY MAP. Candidate representatives characterized by information accessibility and quality for each system context. A template of this workproduct is illustrated in Exhibit C-10.
- DOMAIN CONTEXTS MAP. Contains the set of system contexts to be studied for each representative system. These contexts are linked by a mapping that relates contexts of similar types (e.g., all system in development contexts, system in use contexts, etc.) and indicates their domain-wide relationships.
- PRIORITIZED INFORMATION TYPES. Selected representative systems mapped to a refined list of desired information types to be examined. This workproduct is an annotated refinement of the INFORMATION ACCESSIBILITY — QUALITY MAP. For each representative system, the PRIORITIZED INFORMATION TYPES includes:
  — A list of domain contexts to be examined including scoping by context type (e.g., exclude maintenance environments for all representatives).
  — A list of types of artifacts, stakeholder roles, and processes to be studied in each type of

context.

— An indication of whether data to be acquired is representative, sampled or exhaustive.

## Guidelines

- Find a domain confidante. Although the purpose of *Plan Data Acquisition* is to be systematic about what sources of information are utilized in domain modeling, it is necessary to establish some starting point. In practice, it's useful to identify a ***domain confidante*** who can tell you what artifacts are worth looking at, who is considered an expert within the domain, what other informants to talk to, etc. In some cases, the confidante will also be a sponsor who can facilitate the access to the information sources, but the roles can be distinct as well.

- Representative systems are not necessarily customer contexts. There are good reasons to study systems that may not be candidate receptor systems for implemented assets. Back-filling components into deployed legacy systems that have been in use for some time may be quite difficult. Requirements to back-fill might constrain implemented assets in ways that would frustrate their use in other, more critical contexts. Nevertheless, study of these systems may be quite valuable in deriving a complete and robust descriptive model. By separating selection of systems as information sources from systems as potential customer contexts, the ODM process encourages modelers to examine a rich diversity of data about the domain.

- Stay flexible. Information sources become available at indeterminate times (especially in the case of human informants). Remember that any planned strategy is provisional at best.

## Exit Criteria

- The data acquisition effort for the project has been scoped in terms of representative systems, specific system contexts and types of information sources to be studied.

- Reasonably diverse information sources have been considered in the planning effort. Key trade-offs in artifact analysis and informant interviewing have also been considered.

- Artifacts and informants have been characterized sufficiently to assign resources and set schedules for data acquisition.

## Validation and Verification

- Data Coverage. For every desired information type, there is some representative system with adequate quality and accessibility of data for that type. If this is not the case, either adjust the REPRESENTATIVE SYSTEM SELECTION or adjust the priority of the information type.

- Consistency with resources. The DATA ACQUISITION PLAN is consistent with PROJECT CONSTRAINTS. In particular, the number of representative systems selected is reasonable with respect to project resources.

If these criteria are not met, do a further iteration of the task. If this does not resolve the issues, it may be necessary to reevaluate the DOMAIN SELECTION or, more extremely, the PROJECT OBJECTIVES. Although there is a lot of flexibility in the *Plan Data Acquisition* task, there may be inherent data acquisition barriers that mean certain project objectives will not be met, no matter how thoroughly the *Plan Data Acquisition* activities are performed. These barriers may not become evident until this point in the life cycle. It is much better to acknowledge such obstacles at this point than to proceed further.

*Example.* Consider a domain engineering project within a system maintenance organization that decides to focus on a complex algorithmic domain within the set of systems maintained. It may turn out that the original designers of the legacy algorithms are inaccessible as informants (e.g., in organizations where access is denied or the personnel have retired). Without this DOMAIN INFORMANT KNOWLEDGE, it will not be possible to reengineer legacy code within the selected domain. This conflict should be discovered in the *Plan Data Acquisition* task. Once raised as an issue, it could be dealt with in a number of ways:

— Alternatives to reuse of legacy *code* could be considered, e.g., reuse of test scenarios for regression testing of developer organizations' changes to algorithm implementations. To the extent that a requirement to engineer certain types of assets were incorporated explicitly in the PROJECT OBJECTIVES, this would constitute a modification of the objectives.

— Additional PROJECT RESOURCES could be requested, e.g., engineers with specialized knowledge in technical areas relevant to the algorithmic data.

— The developer organizations where the expertise resides could be approached as new potential ***project stakeholders*** and/or ***domain stakeholders***. Participation in the project would be predicated on discovering sufficient alignment with these organizations' overall ***interests***.

— A different domain of focus could be selected.

There may be resistance to some or all these strategies. For example, focusing on reuse of anything other than code may not feel like "real" reuse within the organization performing the project; the first option might therefore compromise the perceived success criteria for the project. If no strategy can be found to resolve the issue, continuing the project as currently constrained would have an extremely high risk of failure.

## 6.1.2 Examine Artifacts

### Purpose

The *Examine Artifacts* task involves in-depth collection of information to guide development of DESCRIPTIVE MODELS of the domain. In this task we focus on examining and sometimes generating artifacts as input to the modeling process. ***Artifacts*** include primary data about representative systems as well as secondary data such as survey articles or comparative descriptions drawn from textbooks or industry sources.

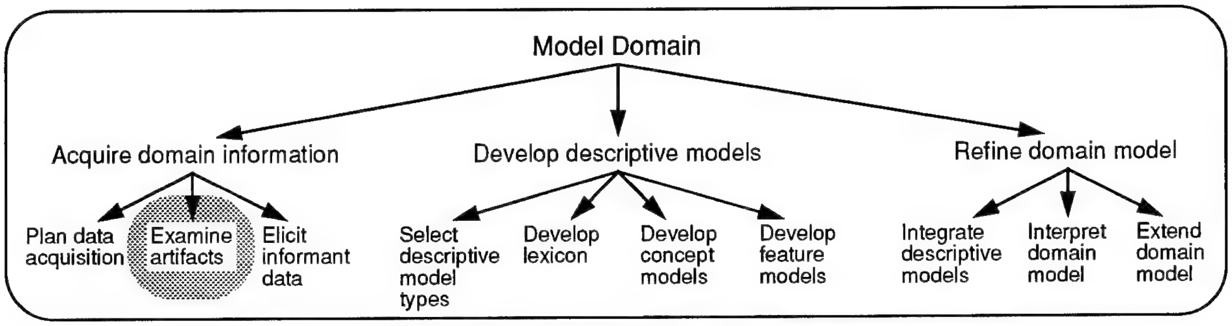


**Exhibit 37.** Examine Artifacts Process Tree

### Entrance Criteria

- The REPRESENTATIVE SYSTEMS SELECTION has been made.
- The DATA ACQUISITION PLAN has been completed with respect to primary artifacts (i.e., those associated with specific representative systems).

These criteria ensure that the right set of artifacts is being examined. If the criteria are not met dual risks will occur in ad hoc artifact analysis:

- resources may be expended inappropriately; and
- information that may strongly influence the models developed will be obtained without a sense of the originating context for the information.

These concerns apply equally to the *Elicit Informant Data* task.

### Inputs

- REPRESENTATIVE SYSTEM ARTIFACTS: The primary inputs to this task.
- DOMAIN ARTIFACTS: Sources of information about the domain not associated with a specific representative system; i.e., not a system workproduct.
- DOMAIN INFORMANT KNOWLEDGE. Commentary on artifacts via comments, documentation, walkthroughs, etc.

### Controls

- DOMAIN DEFINITION. Provides criteria for selecting specific artifacts and filtering relevant data from the artifacts. Also the source of the DOMAIN DEFINITION LEXICON (a component of the INTENSIONAL DOMAIN DEFINITION) refined in this task into the DOMAIN LEXICON.
- REPRESENTATIVE SYSTEMS SELECTION. The set of systems selected from which artifact data will be acquired.
- DATA ACQUISITION PLAN. Overall plan for what types of data to study for each system in the REPRESENTATIVE SYSTEM SELECTION, along with sequencing and team structuring strategies for artifact analysis.
- PROJECT CONSTRAINTS. Controls the resources that can be expended on activities.

  Although these were considered in developing the DATA ACQUISITION PLAN, there will be ongoing decision-making that will depend largely on PROJECT CONSTRAINTS.

### Activities

➤ Form data acquisition teams

Working from the teaming strategies set out in the DATA ACQUISITION PLAN, put the specific teams together. This activity should be performed early in the *Examine Artifacts* task.

➤ Inventory artifacts of interest

Working from the DATA ACQUISITION PLAN, for each system in the REPRESENTATIVE SYSTEM SELECTION, make an inventory of the specific REPRESENTATIVE SYSTEM ARTIFACTS available. The output of this activity is a list of artifacts organized by representative system. This expands the structure of the DATA ACQUISITION PLAN, by inventorying individual information sources rather than categories; a more detailed inventory than was performed in the *Plan Data Acquisition* task. For example, where a DATA ACQUISITION PLAN may call for studying a sampling of software trouble reports from a given system, this is the activity where the inventory of specific reports available is acquired as a basis for selecting the sample.

➤ Select and scope artifacts of interest

Working from the inventory produced in the last activity, and using the DOMAIN DEFINITION for filtering criteria, and the PROJECT CONSTRAINTS for bounding criteria on level of effort, select the particular artifacts to study. The output of this activity is artifact selections from the inventory, possibly prioritized and/or assigned to particular acquisition teams.

Scope the intersection of domain functionality with each representative system. The domain of focus may be both horizontal and vertical with respect to different representative systems, and may be encapsulated or distributed within different representative systems. One system may be a stand-alone implementation of domain functionality, another including it as a discrete component, function or subsystem, a third implementing a distributed subset of functionality.

The key challenges here are applying detailed sampling criteria where required, and determining which artifacts properly belong within the domain of focus. The latter analysis can be non-trivial, especially in cases where the domain does not correspond to a clear structural component of the representative system. For example, this might involve looking through a list of software modules for a large system, determining which modules can appropriately be allocated to the domain scope. In some cases, even individual modules will span the boundary of the domain, with some domain and some extraneous functionality. At this stage, it is better to err on the side of inclusion rather than exclusion.

➤ Access artifacts of interest

Retrieve the actual data in a form that can be readily used for analysis. Log each artifact studied (not necessarily the data itself) in the ARTIFACTS LIST of the DOMAIN DOSSIER. Allocate the artifact, by information type, to the appropriate context as documented within the SYSTEM CONTEXTS MAP for the representative system from which it was derived.

Depending on the support technology involved, this could involve some transformation of data, such as getting on-line access to hard copy documents. There may be some practical obstacles to getting access to certain information, such as issues of classified or proprietary data.

➤ Prepare artifacts for analysis

Based on the selection of artifacts, there are various steps required to get the data into a state suitable for analysis. These steps include:

- Reverse engineering to fill gaps in available data. For example, if architectural diagrams of all representative systems are desired, some may need to be generated from detailed design documents.

- Prototyping artifacts. For example, if only requirements were available for a given system, a prototype architecture diagram could be generated for the purposes of analysis.
- Translating or converting artifacts into common representations/notations. These may be required for the purposes of comparative modeling.
- Developing models of system-in-development contexts. If these contexts are within the scope of the domain and the DATA ACQUISITION PLAN, it may be appropriate to develop models using the process modeling representations and methods of choice for the organization. Quite typically, such models would not have been developed as part of a software engineering effort, unless software engineering was done using a process-driven approach. Even in these cases, the process models might be ***prescriptive*** (what they thought should have occurred) instead of ***descriptive*** (what actually occurred).

This activity results in NEW SYSTEM ARTIFACTS. Some strong caveats apply:

- Perform the activities above only when conditions require it. In particular, generate artifacts for representative systems *only where necessary* for domain engineering objectives.
- If artifacts are developed by project members who are not also members of the application teams for the representative systems in question, they should *not* be considered ***primary artifacts***. (See Guidelines below.)

This activity involves using supporting methods in the following areas, as documented in Section 8.0:

- Reverse Engineering
- System Modeling (for prototyping, engineering into common notations)
- Process Modeling (for modeling system-in-development contexts)

➤ Analyze the artifacts

This is the core activity for this task. Examine the artifacts, looking for key concepts, terminology, characteristic functions and operations in the domain. Record results as appropriate:

— in the DOMAIN DOSSIER when they enrich the model of the representative contexts and their interconnections.

— in the DOMAIN LEXICON, as domain-specific terminology is discovered.

— and in the DOMAIN INFORMAL FEATURES list as each representative artifact is studied and characterized.

The order in which data is examined, teaming strategies, previous background of the data acquirer, integration with *Develop Descriptive Models* activities, all will affect the outcome of the data acquisition process. Iterate back to the Inventory, Select and Access activities as needed.

This activity may involve interaction with supporting methods in the area of Information Acquisition Techniques, as described in Section 8.2.

## Workproducts

■ <u>DOMAIN DOSSIER</u>

- ARTIFACTS LIST. List of artifacts examined, including those called for by the DATA ACQUISITION PLAN and those selected serendipitously. Artifacts can include primary as well as secondary data (i.e., artifacts not tied to an individual representative system).
- For each representative system:
  - A CONTEXT MAP for each context to be studied, with linkage for artifacts, processes, and practitioners active in that context. For example, a design document is linked to the context in which it was produced.
  - A SYSTEM CONTEXTS MAP showing detailed linkage between the contexts, with information and product flow. This information is largely suggested by the context types (e.g., system-in-development, system-in-use contexts) but more details can emerge from data acquisition.
  - INFORMAL FEATURE SETS: Sets of ***informal features*** for the system as a whole, or for particular artifacts. Includes the author of each feature (e.g., a member of the modeling team inspecting a design, a comment noted during a walkthrough with a designer, a comment drawn from a design rationale document).

■ <u>DOMAIN LEXICON</u>

Extends and refines the DOMAIN DEFINITION LEXICON with terms that have specific meanings for practitioners in the system contexts studied (as deduced from the artifacts). The lexicon links domain terms to their occurrence within specific representative system artifacts. Lexicon entries are also added during the *Elicit Informant Data* task. Syntactic relations such as synonyms are noted where they are clearly suggested by the data in the domain. Terms are not standardized in this task. This is done in the *Develop Lexicon* task within the subsequent *Develop Descriptive Models* sub-phase.

■ <u>NEW SYSTEM ARTIFACTS</u>

NEW SYSTEM ARTIFACTS include any of the artifacts for representative systems that were prototyped, reverse or "lateral" engineered to fill gaps in the data. These workproducts can be by-products of the *Acquire Domain Information* task.

Because these artifacts may or may not be clearly associated with a single representative system they are termed only "**system** artifacts". Because the artifacts are created in the course of domain engineering, rather than being existing artifacts that are examined, described, and annotated, they are "**new** system artifacts". Because they are created as data for domain engineering, rather than to build any individual systems, they are considered **artifacts** rather than workproducts.

## Guidelines

- <u>Convert data to facilitate comparison and modeling processes only</u>. Creating new artifacts is *not* a core part of ODM. For example, if it is determined that system designs will all be translated into $\mathrm{IDEF}_0$ diagrams, the motivation should be support of comparative analysis across the artifact, not an organizational decision that $\mathrm{IDEF}_0$ should be the standard design language. The latter motivation can be considered in the DATA ACQUISITION PLAN, but is a separate concern from domain engineering.

This is a significant point of risk, for several reasons:

— If artifact analysis is diverted into system modeling, it can siphon away many resources from the project before the issue is raised. At best, this will satisfy organization needs that are tangential to domain engineering goals. At worst, it may be work that is not valuable as reverse engineering.

— If project members are ill at ease with domain modeling concepts and tasks, there will be a natural tendency to slip back into familiar design activities. For a broader perspective on this issue, see the concluding discussion on Key Challenges at the end of Section 10, Guidelines for Applying ODM.

- Look for unanticipated types of artifacts. Each domain or each system may have artifacts that are only discovered once data acquisition begins. These could include concepts documents, meeting minutes, architecture diagrams, etc. They can be useful sources, particularly for contextual information and rationale. They are less likely to be objects for direct comparative modeling across systems. Formalized versions of these artifacts may become useful types of assets, e.g., requirements checklists.

- Consider data from previous domain engineering efforts. Unless previous domain engineering efforts were performed using ODM or a comparable formal process, system artifacts produced will typically not be linked systematically to a deriving set of representatives. Thus, previous data cannot be allocated definitively to one particular representative system. Previous domain engineering models also may not be attempts to model *variability* across systems in the domain, but may be "commonality only" system models. These models correspond to what common usage would term "generic" models.

- Watch for diminishing returns when looking at analogous artifacts across representative systems. As understanding of the domain increases and an initial model is built, data acquisition becomes increasingly an activity of scanning for new information or exceptions to previous results. Continuing past this point may be a waste of resources. But short-circuiting data acquisition process may result in leaping to conclusions about assumed commonality across the domain, formed from initial data but not validated sufficiently.

## Exit Criteria

Exit criteria for the *Examine Artifacts* task as a whole are determined by the DATA ACQUISITION PLAN. In addition, there is a data acquisition life cycle that can be identified for each artifact and each representative system. Stop gathering descriptive data from specific artifacts when:

- Analogous sets of artifacts have been examined to a comparable level of detail. These will usually span several representatives systems. Different sets of data may have different criteria for completion.

- Key *distinctive* features of representative system artifacts have been identified in terms meaningful to domain practitioners.

  *Example.* Within a given set of representative systems, only one system has a multi-user capability within the domain functionality. As soon as this distinguishing feature has been identified, any further details modeled about that system's multi-user capability serve only to describe the individual system in more detail, not to differentiate it from the other systems. This is a good point to at least pause data acquisition in this area.

- Data from informants becomes necessary to augment and support data derived from the artifacts, e.g., technical data cannot be interpreted further without input from system developers.

### Validation and Verification

Domain data acquired through artifact analysis should be validated carefully. Possible characteristics to consider include the following:

- *accuracy.* Has the data been correctly interpreted? Was it correct to begin with?
- *currency.* Does the data accurately reflect the evolving state of the system concerned?
- *relevance.* Is the DATA ACQUISITION PLAN being followed, and is it proving an effective road map for the kinds of data useful for modeling?
- *level of detail.* Is there a tight filter on the level of detail included in the data incorporated into the DOMAIN DOSSIER? Has data acquisition shifted into system modeling activities?

While the DATA ACQUISITION PLAN will contain broad assessments of these characteristics for different classes of data, the information derived from individual artifacts studied should be validated independently. Cross-validation of artifact data with informant-derived data is essential.

## 6.1.3 Elicit Informant Data

### Purpose

The purpose of the *Elicit Informant Data* task is to utilize a rich and multi-disciplinary set of potential information sources and information-gathering techniques in domain engineering, including:

- published documentation and textbooks,
- system artifacts,
- requirements and market studies,
- interviews with domain experts and developers,
- direct observation of domain practitioners at work,
- team-based expert interviewing and review, and
- instrumented capture of rationale.

### Entrance Criteria

Elicitation of data directly from informants can begin when:

- The DATA ACQUISITION PLAN has selected, prioritized and scheduled informants (e.g., those associated with specific representative systems).
- The DOMAIN INFORMANT MODEL has characterized informants specified in the DATA ACQUISITION PLAN sufficiently to determine an elicitation strategy for those informants.

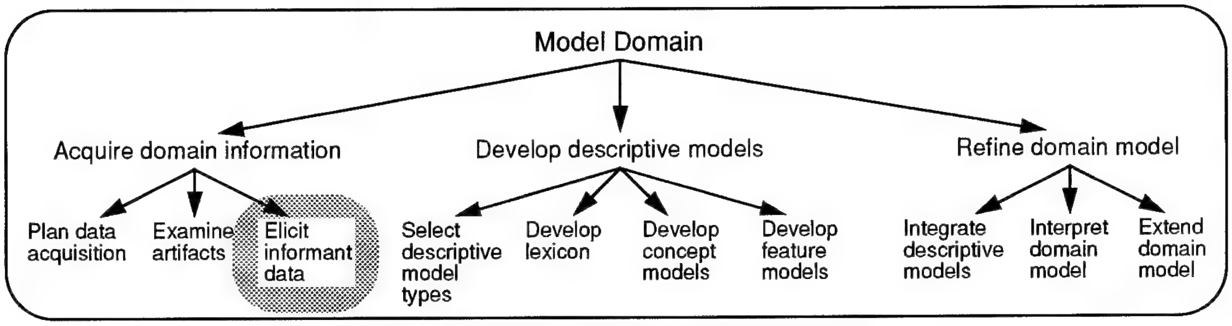


**Exhibit 38.** Elicit Informant Data Process Tree

These criteria ensure that the right group of informants is being interviewed. If these criteria are not met the risks are similar to those described in the Entrance Criteria for the preceding *Examine Artifacts* task. However, the consequences of mistakes are more serious in the case of informants because their time, as well as that of modelers, may be wasted by premature or ill-prepared data gathering efforts. This can compromise the credibility of the project.

## Inputs

- DOMAIN INFORMANT KNOWLEDGE. Knowledge elicited from domain practitioners in interviews or via observation and process capture. Knowledge can include knowledge about specific representative systems or more general knowledge about domain concepts, terminology, etc.

## Controls

- PROJECT CONSTRAINTS. Level of effort must be monitored dynamically for this task.
- DOMAIN DEFINITION. Used as filtering criteria on what information is relevant to the domain; also used for communication of project focus to informants.
- DOMAIN INFORMANT MODEL. Model of informant profiles, used to establish strategies for how to best approach different informants and what kind of information to elicit.
- DATA ACQUISITION PLAN. The overall schedule, teaming strategy, and coordination plan for informant data gathering in association with artifact analysis.

## Activities

Most of the activities for this task depend heavily on supporting methods in the area of Information Acquisition Techniques, as described in Section 8.2. In addition, since this task involves extensive dynamics during interaction with practitioners in the domain, it is useful to review the material on Organization Consulting Skills and Techniques in Section 9.3. The latter skills are considered layers rather than supporting methods because they are useful throughout the ODM life cycle, however, they are of particular importance here.

With the exception of the initial, preparatory procedures, the activities described below are not intended as a sequential list of steps. Instead, they represent a repertoire of techniques that can be applied selectively in this task.

➤ Initial preparation

Many of the activities required for preparation here are identical to the preparatory activities for the *Examine Artifacts* task. They are summarized here:

- Inventory the informants available. This will usually involve a level of detail beyond that in the DATA ACQUISITION PLAN.
- Select the informants and set the guidelines for the form of data acquisition that will be employed.
- Form the interviewing teams.
- Arrange access to the informants. This may involve issues such as cooperation among organizational divisions or separate organizations, funding for the informants time spent in interviews and meetings, etc.

➤ Prepare interviewing materials

This includes preparing any pre-interview information the informants will receive, describing the domain engineering project at the level deemed appropriate for this stage of the project, and describing the project team's expectations for the interview. Indicate if any preparation on the part of the informant is expected, e.g., review of design documents from legacy systems with which the informant was involved. The materials prepared should also include questionnaires, surveys, electronic mail broadcasts, guided question sets for interviews, selected artifacts to be used for walkthroughs, etc.

Some planning for interview follow-up should be done in advance, since some additional demands on the informants' time may be made to review and validate interview results, etc. Any such requests should be made clear before conducting the interview.

➤ Interview Practitioners

Using the characterizations in the DOMAIN INFORMANT MODEL, conduct the interviews, following a protocol designed to produce the kind of information most appropriate for the informants participating.

Factors to consider in conducting the interviews include the following:

- Interview styles (question sets, loose vs. tight facilitation) appropriate for different practitioners, e.g., field technicians, system administrators, expert designers.
- On-site (close to the informant's work context) vs. off-site location.
- Interviewing individuals separately, versus groups of practitioners. For group interviews, drawing together cross-functional groups from same project, or practitioners who play analogous roles on different projects.
- Conducting interviews with individual or data acquisition groups from the project team. Composition of interviewing teams in terms of mix of domain and modeling expertise, etc.
- Possible interview protocols to use:
  - Scenario-based elicitation of work process (talking through a task);

- — walk-throughs/inspections of artifacts in interview settings;
- — soliciting "war stories" about past project experience; or
- — requesting direct comparative evaluations from expert informants (e.g., people with experience on multiple systems).

For the purposes of domain modeling, one way to distill data in the acquisition process is to summarize it in INFORMAL FEATURES. These are free-text statements about domain functionality and initial insights into differentiators as gleaned from informant data and language. Informant interviews with the specific objective of eliciting or refining INFORMAL FEATURES can be conducted in various ways:

- as structured walkthroughs of exemplar artifacts;
- as an interview following a questionnaire format, where questions may be derived from other feature extraction processes by modelers; or
- as an interview that follows the thread of the informant's experience, either historically (i.e., on a project-by-project basis) or in terms of typical work processes encountered in the informant's role in domain-specific development.

➤ Observe practitioners in context

Direct observation of work practice in the domain can be a powerful adjunct elicitation technique to interviewing. Social scientists are familiar with what is known as the "say vs. do" dilemma. What people report that they do often bears faint resemblance to what they do in practice. Try to directly observe the core work practices that characterize the domain. For the system-in-use context, this could include watching system operators perform domain functions using fielded systems. One purpose for observation of system operators would be to discover needed domain operations that are performed in the use context, e.g., work-arounds to system bugs or limitations, aggregate series of operations that are not directly supported by the system, etc.

For system-in-development contexts direct observation could include sitting in on design review meetings. The purpose might be to elicit developer processes (e.g., testing routines, repetitive programming) that could be supported with reuse at the process level.

➤ Instrument environments for data collection

An alternative or adjunct to the observation strategy above is to use automated support to derive instrumented data that includes process information about work practice in the domain. This has the advantage of being less intrusive than a human interviewer in the context. On the other hand, data derived in this fashion must be interpreted care precisely because the contextual information obtained in face-to-face interactions is missing.

➤ Participate directly

A final strategy to consider is to participate directly in domain work processes. This includes participation in development or use of domain functions. In either case, participant observation skills are needed to derive data simultaneously as the work is performed.

*Example.* In performing the Outlining domain modeling prototype, the modeler got access to several commercial outlining programs and used them extensively to record the project data. Use of several different programs highlighted common and variant features in the implementation, terminology, and assumed contexts of use for the various programs. It would have been very difficult to derive this data originally from either artifact analysis or "expert" interviews, although once certain insights were obtained, it was relatively easy to validate them with supporting data.

## Workproducts

■ <u>DOMAIN DOSSIER</u>

The evolving "map" of the data sources from which the DESCRIPTIVE MODELS are derived. In this task, the dossier is extended to include:

- Reports from interviews with practitioners, observation of work practices, instrumented process capture, etc.
- Informal features elicited from informants.

■ <u>DOMAIN LEXICON</u>

Extends and refines the DOMAIN DEFINITION LEXICON with terms that have specific meanings for practitioners in the system contexts studied as deduced from informant data. The lexicon links domain terms to their occurrence as recorded during interviews with specific informants, as a supplement to the linkage to particular representative systems and artifacts in the previous task, *Examine Artifacts*. Syntactic relations such as synonyms are noted where they are clearly suggested by informants.

## Guidelines

- <u>Just learning about the domain is not modeling</u>. There is a fine line between data acquisition for the purposes of modeling and for the purpose of familiarization with the domain. The domain engineering effort can be compromised by allowing this distinction to blur. This is a danger throughout the *Acquire Domain Information* sub-phase, but particularly in the *Elicit Informant Data* task, where there is face-to-face interaction involved.

  Facilitating the education of trainees in the domain can be a legitimate project objective for the DOMAIN MODEL. Even in this case, though, domain modelers must do more than educate themselves about the domain. Modelers may spend afternoons drawing on a blackboard with a domain expert, and emerge with a satisfied feeling of personally understanding the domain much better, without having necessarily furthered the domain engineering process. Great effort can be expended in such activity, which often goes unmeasured and unmanaged. This can also obscure how much time has really been demanded of informants.

  Clear documentation of the intended information-gathering process in the DATA ACQUISITION PLAN and the DOMAIN INFORMANT MODEL is one technique to mitigate this risk. This formality is intended to heighten awareness of those situations in which domain information is being transferred. However, formal planning will be of little use without constant attention to this issue.

- <u>Treat knowledgeable team members as informants</u>. If the domain engineering team includes domain experts it is important to document them explicitly as informants in the DATA ACQUISITION PLAN. The temptation will be to treat team members' domain knowledge informally. This will undermine the value of the DATA ACQUISITION PLAN as an accounting of the empirical basis for the final DOMAIN MODEL produced in the *Model Domain* phase, since informal and unverified domain knowledge will be incorporated into the model.

Having team members who can serve as real informants is a valuable resource in many ways. Other team members can practice informant interviewing with fellow team members in a low-risk setting. Interviewing teams can be strategically configured to include some members with domain expertise, to establish credibility with informants and allow for rapid information transfer. But utilizing such resources effectively requires an accurate and impartial assessment of the team members' status as informants.

- Maintain focus. As with artifact analysis, be careful when eliciting informant data to stay focused on data that supports comparative modeling. One of the dangers is that in eliciting scenarios of work process the domain focus will be lost. Domain boundaries may not be evident or even intuitively obvious to informants. Informants probably don't cluster their experience along these lines. Facilitate and guide the interview format to keep focused on data relevant to the domain. Otherwise a lot of extraneous data may need to be filtered, more painfully, in later transcription and processing of the interview data.

- Treat informants as potential customers. The ultimate goal of domain engineering is to develop an asset base that will get used. All contacts with informants, though ostensibly focused on information gathering, also serve to convey information about the domain engineering project, modelers' competencies and knowledge of the domain, and the nature of domain engineering itself.

  Paradoxically the best way of "selling" the project at this stage is to treat the informant as a valued source of information, rather than performing an oblique selling job on the asset base that will be built. People who feel they've had input into the content of domain models are more likely to feel a sense of ownership when asked to use assets developed on the basis of those models.

- Respect confidentiality. As with any data collection in organizations, some domain information will be sensitive for various informants. In the domain engineering context data is being collected specifically for comparative purposes. This may raise issues such as risks of the data being used for performance evaluation or inter-divisional competition.

  ODM encourages maintenance of traceability information where feasible. Being able to trace data back to specific originating contexts is important in developing the interpretive domain model. Unfortunately, this also makes anonymity difficult to guarantee. This is only one aspect of a larger set of complex ethical issues. At a minimum, any commitments made to informants must be respected by the modeling team, if the project is to maintain credibility within the organization.

- Pay attention to resistance. In domain engineering practitioners are asked to share their expertise. Reluctance on the part of informants to provide some information may be due to a number of factors. For example, disclosure of expert knowledge may be seen as a threat to job security. Be alert for resistance, and respect informants' concerns and their rights to their knowledge.

- Leave some stones unturned. In each individual interview, and in the elicitation process as a whole, it is critical to be attentive for the perceived point of diminishing returns. If the modelers sense the interview has reached this point the informant may have concluded the same thing some time earlier. Rather than risk exhausting informants' good will or making them feel their time is wasted, it may be preferable to intentionally leave gaps that can be filled in during later sessions, when DESCRIPTIVE MODELS will be presented for validation and interpretation by informants.

## Exit Criteria

- Key informants called out in the DATA ACQUISITION PLAN have been interviewed. At a minimum, these should include representative informants for each major domain practitioner's role within the domain contexts of interest.
- Sufficient data has been elicited from informants to adequately describe domain terminology used by practitioners, and to document differentiating features of representative systems for domain practitioners.

As discussed in the Sequencing sub-section for the *Acquire Domain Information* sub-phase, artifact analysis and data gathering from informants are closely interleaved activities. Therefore, the exit criteria for this task and the previous task, *Examine Artifacts*, focus on the transition from data acquisition in general to the activities in the next sub-phase, *Develop Descriptive Models*.

## Validation and Verification

The *Elicit Informant Data* task as a whole serves a validation function with respect to the information acquired through artifact analysis. Informant data can be validated as well with respect to data derived from artifact analysis, and/or using any of the following means:

- Validate via follow-up review. Go back to the informants with the data that has been captured. Check for accuracy, as well as to verify that you are using the terminology of the domain and not requiring informants to adopt language introduced arbitrarily by modelers.
- Validate via other informants. Cross-validate information of informants who are practitioners in the same system contexts; e.g., the designers and testing engineers for the same systems. Cross-validate informants who perform analogous roles in different system contexts; e.g., testing engineers from different representative systems. Validation can be performed through collation of results obtained from multiple individual interviews, or by bringing informants together for focus-group style information gathering sessions.
- Validate via other means of data acquisition. Consider the full range of information gathering techniques described in the activities for this task, and the supporting methods for Information Acquisition described in Section 8. For example, augment interviews by directly observing, participating, or instrumenting systems to derive documentation of actual practice.

## 6.2 Develop Descriptive Models

### Purpose

In the previous sub-phase, *Acquire Domain Information*, a rich set of data has been gathered for a set of representative systems in the domain. The primary purpose of the *Develop Descriptive Models* sub-phase is to distill this information into formal models that represent the ***commonality*** and ***variability*** across the representative systems studied. This picture of the differences between systems provides an essential intermediary form that derives the full benefit of studying multiple systems in parallel. It helps in the uncovering of rationale and context in the system artifacts, the further task which is the focus of the subsequent *Refine Domain Model* sub-phase. Understanding this rationale will be essential in the eventual goal of reengineering artifacts to be reusable in a broader set of application contexts.

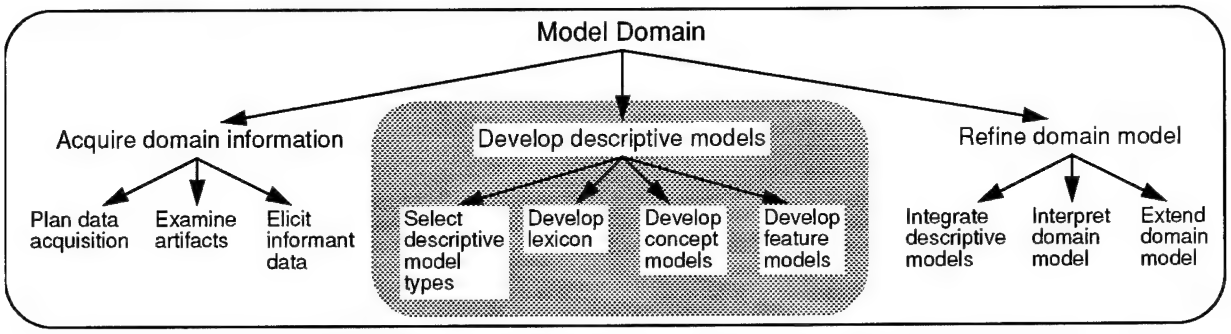


**Exhibit 39.** Develop Descriptive Models Process Tree

### Description

In the *Develop Descriptive Models* sub-phase, modelers develop a number of separate models selected and tailored to suit the domain of focus, ORGANIZATION CONTEXT and domain engineering PROJECT OBJECTIVES. This sub-phase focuses on ***descriptive modeling***, which is both a process observing certain formal constraints, and a particular approach or mindset somewhat different than the traditional problem-solving approach in engineering. Formally, terms, concepts, and features described in the DESCRIPTIVE MODELS are justified by the ***precedent*** of occurring in at least one system or system requirement included in the REPRESENTATIVE SYSTEMS SELECTION. There is no implicit commitment to provide assets to support any particular context studied in this sub-phase. In ODM, these formal guidelines are used in part as techniques to help reinforce the attitudinal shift to descriptive analysis.

Domain modeling can be thought of as defining a formal language for describing domain entities and behavior. Based on the analogy to a language, the lexicon provides the *syntax* for this language, the concept models provide the *semantics*, and features correspond to actual *sentences* or *statements* in the language. Unlike a natural language, however, the domain language produced in domain modeling creates a fixed repertoire of statements that can be made about domain entities. The ***feature profile*** of an artifact (or later, an asset) is the set of these feature statements that hold for the artifact or asset in question.

As shown in Exhibit 40, the *Develop Descriptive Model* sub-phase consists of four tasks.

- In the *Select Descriptive Model Types* task, the set of descriptive models to be built is deter-

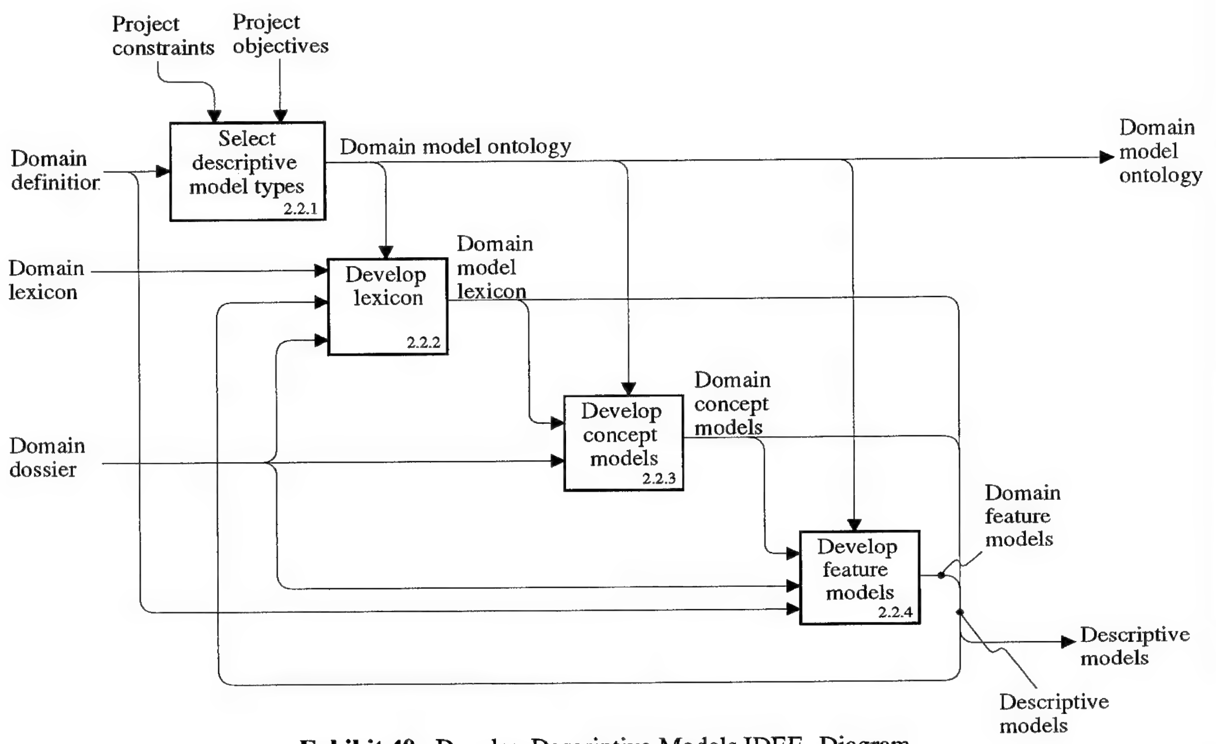


**Exhibit 40.** Develop Descriptive Models $IDEF_0$ Diagram

mined and their interconnections are modeled.

- In the *Develop Lexicon* task, modelers document and manage domain terminology reflected in the artifacts and transcripts of informants' terminology. By normalizing the syntactic domain terms in the DOMAIN MODEL LEXICON, the descriptive models can be focused on essential semantic relationships. Lexicon development is considered part of modeling rather than data acquisition because a certain amount of comparative modeling is required to identify synonymous terms and select an appropriate standard term to use in the DESCRIPTIVE MODELS.
- In the *Develop Concept Models* task, lexicon terms are mapped into DOMAIN CONCEPT MODELS that define semantic relationships among domain terms. Separate models created in this task are validated for internal and cross-consistency, and for completeness and conciseness with respect to domain information.
- Concept models provide the groundwork for the *Develop Feature Models* task, in which models of domain ***features*** are developed, which differentiate domain systems and artifacts.

## Sequencing

- Use the *Develop Lexicon* task as a filter for domain terminology from the DOMAIN DOSSIER into the DESCRIPTIVE MODELS. Developing the lexicon separately and as a pre-cursor to semantic modeling means the lexicon will be more likely to catch important domain terminology that might not fit within the set of models being developed; this is a good completeness check for the modeling process as a whole. Also, the DOMAIN MODEL LEXICON

Modelers searching for object serves as a bridge back to domain informants, who do not need to know the semantics of the taxonomic modeling representation chosen for the DOMAIN CONCEPT MODELS and DOMAIN FEATURE MODELS in order to interpret and validate the lexicon.

- The *Develop Concept Models* and *Develop Feature Models* tasks can be performed in many sequences. It may prove easier to work forward from the lexicon to concepts and then on to features; or to work backwards from features (derived from INFORMAL FEATURES), defining only the concepts needed.
  - The former strategy may yield finer-grained feature models, but may also get off target and fail to identify the features of critical interest to domain informants.
  - The latter, top-down strategy can result in a more concise concept models, which grow only as needed to "cover" the specified features. The risk is that some important underlying concepts will not be identified. Consequences may only show up in subtle ways later on in modeling, where the model may lack a certain robustness under transformations.
- Regardless of the sequencing strategy followed, all three levels —lexicon, concept and feature models—are important for the robustness and completeness of domain modeling.

## 6.2.1 Select Descriptive Model Types

### Purpose

To support ODM's overall objectives of sufficient formality to support parallel restructuring of legacy artifacts into reusable assets, the *Develop Descriptive Models* sub-phase consists of the development of a number of individual models addressing various aspects of the domain and levels of abstraction. In the *Select Descriptive Model Types* task, modelers scope the specific set of DESCRIPTIVE MODELS to be built. The set of models is formalized in the DOMAIN MODEL ONTOLOGY, which documents the ***domain-specific meta-model***, or "model of models" for the domain. It specifies the scope for each model, and relationships and interconnections between the models. Modelers determine which domain models to build, the level of detail at which to elaborate the models, and any necessary interconnections between the models.

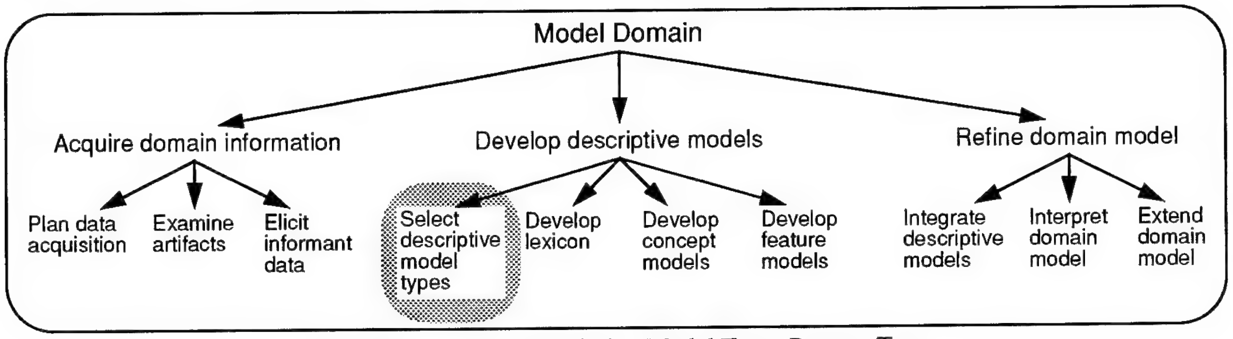


**Exhibit 41.** Select Descriptive Model Types Process Tree

The ***ontology*** of each model determines the types of entities that are the subject of the model. These may range from concrete artifacts, such as code modules or other system artifacts, to more abstract conceptual entities such as types of errors, objects, or operations in the domain. Each model has an associated ***root category name*** which is the effective root for the taxonomy to be built.

The Domain Ontology Model provides the following benefits in the life cycle:

- Ensures consistency in interpretation. The need to explicitly define the ontology of a model reflects the experience of researchers in knowledge representation, where subtle differences in assumptions about what a given model is talking about can lead to confusion.

  *Example*. One model might define Wines in terms of wine regions (e.g., Bordeaux); another model might use the same term as the name of a generic category of wines, where particular vintages would be the individuals. Still another model could use the term to refer to individual bottles from a single vintage (e.g., for a quality control group). In each case, although the general subject is wine, the specific semantics of what an individual in the model stands for is different.

  In domain modeling, where multiple models may be developed by multiple modelers, lack of explicit shared concepts about the purpose and content of models can result in models of poor quality, inconsistent semantics and lack of modularity. While detailed connections between different models are not worked through during the *Select Descriptive Models* task, the general relationships should be made clear.

- Provides additional scoping of modeling effort. Domain modeling, by its nature, is an activity difficult to bound on the basis of model content. A system model is complete when a system can be implemented from the model. But when is a domain model complete? Deriving appropriate exit criteria for the *Develop Descriptive Models* sub-phase is thus a central issue in domain engineering. Criteria are required for scoping not only data to be acquired but the models to be derived from that data.

- Facilitates discovery of domain-specific model types. Each domain includes unique areas of knowledge requiring unique models; i.e., not only the *content* but the *type* of the model is domain-specific. While such models can be loosely classified according to a standard taxonomy, they have important distinctive qualities that must be considered as part of the modeling effort. In fact, these models will often be the most crucial for the domain engineering effort.

- Coordinates independent modeling activities. In any but the smallest projects, modeling may involve multiple people working in parallel. Unlike system development, where modular decomposition is designed to support independent development of units, domain models tend to have numerous relationships and interconnections. Without an overall structure linking the models to be built, it is all to easy to work at cross-purposes.

- Provides rationale for information source selection. Knowledge of what models to build can inform decisions about what kind of data to collect; what information is extracted and traced from artifacts (via keyword search, highlighting, etc.) and what kinds of questions are asked in interviews.

## Entrance Criteria

The *Select Descriptive Model Types* task can begin when:

- Domain of focus has been selected. The model types to be selected are dependent on the domain of focus.

- Domain of focus has been bounded. The *Bound Domain* task has produced an initial INTENSIONAL DOMAIN DEFINITION and EXEMPLAR SYSTEMS SELECTION and these workproducts have been cross-validated for consistency. The set of model types to be selected will further restrict this domain boundary.

- Key interfaces to domain functionality have been identified in the DOMAIN INTERCONNECTION MODEL.

Initiating this task before the above criteria are met raises several areas of risk for the *Develop Descriptive Models* sub-phase. Models unnecessary to the domain may be developed. Key types of models, specific to the domain, may not be considered in the overall structure and relationships of the DESCRIPTIVE MODELS, and may be missed entirely or added later in an ad hoc manner.

## Inputs

- DOMAIN DEFINITION. Defining features become the roots of individual feature models.
  - DOMAIN INTERCONNECTION MODEL. Structurally related domains may require interface descriptions that cover a specified range of variability.

## Controls

- PROJECT OBJECTIVES. Used to develop criteria or specific candidates for descriptive models of intrinsic interest and potential assets of interest.

  Domain Selection Rationale included may indicate potential assets of interest.

- PROJECT CONSTRAINTS. Constrains the extent of descriptive modeling, based on available resources, skill levels of the team, etc.

## Activities

➤ Map potential assets of interest to descriptive models

Scan the PROJECT OBJECTIVES, particularly the DOMAIN-SPECIFIC PROJECT OBJECTIVES, looking for any objectives that call for particular types of assets to be built. Do some "backwards reasoning" from these potential assets of interest to the descriptive models necessary to develop these assets. Validate these in light of the list of information sources to be examined. There will not be a one-to-one mapping of artifacts to be studied and model types per se. This step will help to determine if the right information is being collected for the desired set of models.

For example, suppose that for the domain selected there would be strong interest in a software requirements checklist. This will mean that:

1) requirements data should be examined, and

2) a descriptive model of the commonality and variability in requirements documents structure, etc. should be developed.

The modeling team is not committed to building all (or any) of the assets used in the rationale for given descriptive models in the set.

➤ Determine DESCRIPTIVE MODEL types

Work from the DOMAIN DEFINITION to derive other required model types. Each defining feature will become the root of a DESCRIPTIVE MODEL expressive enough to capture the significant variation across the representative set with respect to the broad feature category. In addition, interfaces to structurally related domains will need to be covered by some descriptive model to accommodate the variability in the interface

➤ Select descriptive model types to develop

The previous steps have generated candidate descriptive models to develop based on various rationale. In this step we look at the descriptive model set as a whole and make final selections.

This might include consolidating certain models into one, eliminating certain models as overlapping or being potentially too redundant.

This step might also involve deciding that some other models are necessary for the balance of the set as a whole. For example, suppose that project objectives call for a model of requirements and code artifacts. Designs are not as high a priority, because most of the legacy systems used a different design representation than that being adopted for new systems. In this validation step, we may decide that it will still be necessary to model the design phase of the life cycle to achieve sufficient traceability from requirements to implementations.

➤ Select the representation method

Select the modeling representation method in which to formalize the list of selected descriptive model types to develop. While ODM does not entirely constrain the choice of this method, we recommend that the DOMAIN MODEL ONTOLOGY be represented using a ***taxonomic modeling*** supporting method. For information about ***taxonomic modeling*** supporting methods see Section 8.3.

There are several reasons for the taxonomic modeling recommendation. The DOMAIN MODEL ONTOLOGY will be integrated with the DESCRIPTIVE MODELS in the *Integrate Descriptive Models* task of the *Refine Domain Model* sub-phase. This suggests use of the same method for the DOMAIN MODEL ONTOLOGY and the DESCRIPTIVE MODELS. In addition, the DOMAIN MODEL ONTOLOGY may be refined by including linkage to ***starter models***. This restricts the supporting methods that are most suitable, as they must have sufficient representational power to support ***meta-models***, models containing entities referring to other models.

➤ Develop DOMAIN MODEL ONTOLOGY

The list of potential descriptive models to be developed is refined into an integrated framework within which each proposed model has only a "root" node as a placeholder. The idea is to establish some loose semantic relations between the various models as early as possible.

➤ Develop Modeling Conventions and Coordination Plans

To the extent that conventions are needed in modeling that extend the requirements of the modeling representations and methods, these are established here. Coordination plans include ways of ensuring that separately developed models will be able to be integrated.

## Workproducts

■ DOMAIN MODEL ONTOLOGY

- DESCRIPTIVE MODEL TYPES. A list of DESCRIPTIVE MODELS to be developed. Used by modelers in allocating modeling tasks, filtering data, and model development. Each model type includes the following attributes:

— Name of model to be developed;

— Derivation/classification of model type (general, domain-specific);

— Types of entities that are the focus of the model;

— Rationale for why the model will be developed.

- DOMAIN MODEL INTERCONNECTION MAP. A ***meta-model*** that shows the relationships between the model types that have been chosen for descriptive modeling.

  *Example*. For the Outlining domain, we decide that we will build a model of the outline data structures themselves; i.e., the document structure that includes headings at various levels, heading text, etc. In the DESCRIPTIVE MODEL TYPES list, we:

  — term this the "Outline Document Structure" model.

  — classify it as a type of ***object model*** within the set of DESCRIPTIVE MODELS we expect to develop.

  — define it as "the model describing variations in the structure of outline documents and in the static components of these documents".

  — document the rationale: we intend to produce an "outline interchange format" specification that can be used to transfer outline data across applications as one output of the domain engineering effort.

## Guidelines

- Look for immediately useful models. Consider models that may have intrinsic, immediate value to the organization, independent of their place in the development cycle for reusable assets. DOMAIN MODELS themselves can be ASSETS if they improve software development process in some tangible way. Where DOMAIN MODEL value will only be obtained if considerable resources are expended, in excess of those required for the purposes of domain engineering, this strategy may involve some trade-offs. The domain engineering objectives should be kept in mind at all times, or the project risks diluted or overloaded objectives. This rationale for building models does not commit the modeling team to completing DESCRIPTIVE MODELS to the degree of detail necessary to fulfill its value to the organization.

- Allow for redundancy in models. Avoiding inadvertent redundancy in modeling is a primary purpose of the *Select Descriptive Model Types* task. But modelers may decide to deliberately model some data redundantly in multiple models for several reasons:

  — to verify modeling data by independent/parallel development.

  — to generate alternative views of the model entities for different audience/uses.

  — to assist in integration of models at a later phase.

  The possibility of generating multiple views from a single model representation should also be considered.

- Document model boundary issues and decisions. Negotiating clear boundaries between the scope of different models will be an ongoing challenge in domain modeling. These issues are analogous in some respects to overall boundary issues for the domain explored in the DEFINE DOMAIN process. As boundary issues are encountered and resolved, it is good practice to document the process and rationale behind the resolution of the issues. If model boundaries change maintaining consistency in data already modeled will also require traceability support.

### Exit Criteria

- A reasonable set of model types has been specified. The set need not be comprehensive. In fact, erring on the side of economy is probably a good idea. However, there should be several distinct models specified.
- The ontology of each model is defined. There is clear documentation of what is and is not included in the scope of each model. Potential areas of overlap between models have been addressed. A supporting method for representing the ontology has been selected and used.

  Relationships between models can remain informal as subsequent tasks are initiated. The relationships can be formalized incrementally as modeling proceeds.

### Validation and Verification

- Validate the set of descriptive models for completeness and economy with respect to the set of suggested descriptive model types outlined in the other task descriptions for this subphase. Understand rationale for inclusion or exclusion of these suggested types for the domain of focus.
- Formally validate the DOMAIN MODEL ONTOLOGY using the resources of the supporting method selected.
- Validating common understanding of model types. To validate that the members of the modeling team have a common understanding of the distinctions between the various model types specified, take a small set of sample data and have modelers independently allocate the data to the appropriate models. Resolve conflicts in allocations to ensure that the formal descriptions of the model types are consistent and understandable.

## 6.2.2 Develop Lexicon

### Purpose

The DOMAIN MODEL LEXICON collates terms drawn from the domain, annotated to reflect information about each term with domain-specific meaning, such as lexical relationships between terms (synonyms, antonyms, oppositional pairs and inverse operations, e.g., "promote" vs. "demote" headings in the Outlining domain).

Terms can be drawn from all domain information sources, including development artifacts (code, design, tests), documentation, literature, and informal terminology noted in interviews and meetings with domain experts. Information sources from the DOMAIN DOSSIER are ***parsed*** in order to generate data for the lexicon.

The DOMAIN MODEL LEXICON provides:

- A description of key domain terms independent of the taxonomic modeling representations used to develop the DESCRIPTIVE MODELS. This facilitates communication with informants

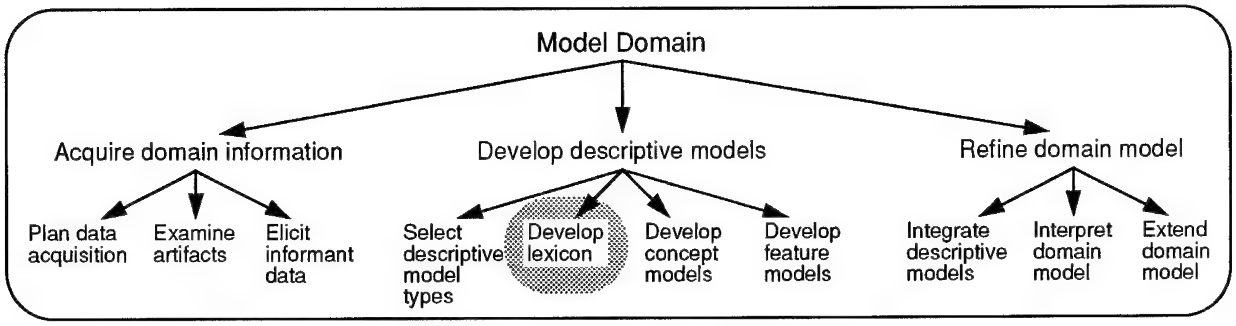


**Exhibit 42.** Develop Lexicon Process Tree

who may not be familiar with these taxonomic representations. Glossaries and data dictionaries are documents with which many technical workers are comfortable. The lexicon may therefore provide an early bridge to domain informants otherwise difficult to draw into the modeling process.

- A mapping between different ***language communities*** to provide practitioners in these communities (e.g., different developer or user environments) with a basis for consistent communication. This terminological cross-translation may itself be a valuable outcome of domain engineering.
- A controlled vocabulary filter to the model-building process, so that syntactic variation does not complicate the semantics in the models.

## Entrance Criteria

- The REPRESENTATIVE SYSTEMS SELECTION has been made. Lexicon terms are identified throughout the domain engineering life cycle. However, the specific role of the DOMAIN MODEL LEXICON as a filter for controlling vocabulary only becomes operative once representative systems have been selected. These are the distinct "language communities" that will be coordinated through use of the DOMAIN MODEL LEXICON.

The DOMAIN MODEL ONTOLOGY need not be complete before this task begins.

## Inputs

- DOMAIN DOSSIER. Used as the primary source for lexicon terms.
- DOMAIN LEXICON. This is the initial lexicon produced in the *Define Domain* sub-phase of the *Plan Domain Engineering* phase. It is refined and extended in this task.
- DESCRIPTIVE MODELS. New terms introduced in the DOMAIN CONCEPT MODELS and DOMAIN FEATURE MODELS are fed back into the DOMAIN MODEL LEXICON.

## Controls

- DOMAIN MODEL ONTOLOGY. Indicates the priorities of what models are to be developed from lexicon data. This is not a hard filter on what is admitted to the lexicon. Allowance is made for domain terms to be included in the lexicon that do not clearly map to selected model types.

## Activities

➤ Select supporting methods for lexical analysis

Most domain analysis methods advocate gathering domain terminology in some consolidated form. Perhaps the most thoroughly elaborated method in this respect is DAPM [Prie91a], which borrows lexical analysis techniques from the field of library and information science and includes a detailed process model for lexicon creation. The lexicon can also be derived in part from data dictionaries created as part of various structured analysis and object-oriented methodologies.

➤ Record modelers' informal domain vocabulary

If domain modelers are knowledgeable in the domain, they may enter the analysis phase with a fairly large repertoire of existing terms as part of their informal knowledge. In this case, it may be useful to first do a prototype lexicon based solely on this informal knowledge, to rapidly capture anticipated terminology without explicit references to information sources.

This prototype lexicon can be created as soon as the domain is selected, and later cross-referenced to tracked information sources. Prototype terms should be uniquely distinguished from later additions to the lexicon. In addition to providing a quick entry to the modeling process, the prototype lexicon can provide useful process documentation. For example, prototype terms eventually excluded from the lexicon, and lexicon terms that were not originally included in the prototype, show the level of domain knowledge within the team at the start of the modeling effort.

➤ Identify terms within DOMAIN DOSSIER

Highlight and delimit terms within dossier materials used in descriptions of domain functionality. For data elicited from informants, key terms should have been noted in the interview report. For artifacts, some analysis will generally be required. Working on parallel artifacts across representative systems will help make variant usage of terms more evident.

➤ Find precedent for initial terms

Look for precedent in the dossier for terms elicited in the prototype lexicon. This can be done in parallel with the scan of the dossier materials.

➤ Update lexicon

Add new terms to the lexicon, with linkage to their *precedence* within the DOMAIN DOSSIER. If duplicate terms are identified, and the usage appears consistent, it is not necessary to annotate the lexicon with multiple occurrences of the term, except as it clarifies usage across the representative systems set. If the usage appears inconsistent, the different usage of the term should be flagged as a "homonym".

Write brief definitions for the terms. Highlight use of other lexicon terms within definitions. Terms can include phrases as well as single words. For example, in the Outlining domain "heading text" is a compound term used with the intended consistency of a single term.

➤ Cluster synonymous terms

Associate synonymous terms as a ***term cluster***. Conventional synonyms, acronyms, case, voice (e.g., active-passive) and plurality variants, etc., may be clustered together. For example, the terms *orient domain* and *domain orientation* should be in the same term cluster.

➤ Select canonic term for each cluster

For each term cluster, select a ***canonic term***. This is a standard term to be used in normalized references in other lexicon term definitions in the term cluster. This standard term is also the term used in the DESCRIPTIVE MODELS.

➤ Identify other syntactic relations between terms

Relations may include homonyms (same word with different meanings) and antonyms (i.e., mutually exclusive terms). Don't attempt to do deep semantic modeling in the lexicon.

➤ Convert features to canonic terms

As a validation step, and also in preparation for the formal modeling of the subsequent *Develop Concept Models* and *Develop Feature Models* tasks, convert INFORMAL FEATURES derived from informant data or artifact analysis into statements using canonic terms. This is a good way to find out quickly how comprehensive the lexicon is (and is not!). In addition to working from the INFORMAL FEATURES, go back to the DOMAIN DEFINING FEATURES within the INTENSIONAL DOMAIN DEFINITION and convert these features to canonic lexicon terms as well.

➤ Extend lexicon to include terms selected by modelers

As modeling proceeds, iteratively update the lexicon to serve as an ongoing snapshot of the modeling team's relative consensus on domain language. In this way, the DOMAIN MODEL LEXICON remains a current snapshot of all the vocabulary introduced in the model, including terms borrowed from domain practitioners and those new terms introduced by modelers to express subtle differentiations not required in informal practice.

## Workproducts

■ DOMAIN MODEL LEXICON

Contains the canonic terms from the DOMAIN LEXICON for each concept requiring differentiation in the domain. The DOMAIN MODEL LEXICON explains domain modelers' terms to the community. Each canonic term is mapped to other domain lexicon terms. Documented rationale for the choice of canonic terms is optional.

## Guidelines

- Don't attempt to model semantics in the lexicon. It is easy to blur the distinction between the lexicon structure and the deeper semantic modeling of concept and feature modeling activities. However, this will compromise the primary usefulness of the lexicon as a front end to these other models, and a means for communication with domain practitioners who have no background in the modeling formalism used by the project.

  Some terms chosen by modelers may be filtered into the lexicon. For example, concepts that represent subtle distinctions for which there is no clear terminology in the domain. If the lexicon is updated to reflect taxonomic relationships, don't allow this to alter its usefulness as a flat term space. A reader should not need to understand modeling semantics to find terms in the lexicon.

- Reflect practitioners' terminology. Semantic relationships recorded in the lexicon should be inferable from information sources, along with the terms themselves, rather than a second level of interpretation provided by the domain modelers. The lexicon should be confined to domain practitioners' language and relationships they perceive as significant. New relationships perceived by modelers should be noted informally and separately.

## Exit Criteria

The DOMAIN MODEL LEXICON does not need to be complete for subsequent model development tasks to proceed. It serves as a pipeline between artifact and informant data in the DOMAIN DOSSIER and the DESCRIPTIVE MODELS. Maintaining the DOMAIN MODEL LEXICON is therefore an ongoing task throughout the *Model Domain* phase. Nevertheless, an initial version of the lexicon should be stabilized early in the *Develop Descriptive Models* sub-phase. This initial phase of the *Develop Lexicon* task can be considered finished when:

- The Domain Model Lexicon has captured key terminology needed to facilitate data acquisition from artifacts and informants from multiple representative systems.
- Feedback from informants has been obtained. The lexicon should be comprehensible to informants without their needing to understand the modeling representation formalism.

## Validation and Verification

- Formally validate the DOMAIN MODEL LEXICON by applying the following criteria:
    - *Minimality.* All terms included in the lexicon have specific meaning within the context of the domain. No general definitions of terms are intermixed with domain terms.
    - *Compactness.* All synonyms within the lexicon have been identified and grouped within a single term cluster. No two canonic terms that are synonyms.
    - *Coverage of models.* When updated mater in the modeling process, the lexicon contains all new terms introduced by modelers within the DESCRIPTIVE MODELS.
- Convert informal features to canonic terms. One way to validate the soundness of choices for canonic terms is to take informal feature descriptions from informant data and convert them to statements that utilize the canonic terms. Verify that the canonic terms are a rich enough set to convey the needed sets of distinctions made by domain practitioners. Be alert for terms occurring in the data that do not map clearly to a canonic lexicon term, but seem to have a distinct semantic sense within the domain. Add these to the DOMAIN MODEL LEXICON as needed. This is considered a validation activity within this task. Later in the *Model Domain* phase informal features will be more systematically converted into canonic lexical form.

## 6.2.3 Develop Concept Models

### Purpose

DOMAIN CONCEPT MODELS describe abstract concepts or entities, primarily within the operational environment for the systems within the domain. Modelers assign canonic terms from the DOMAIN MODEL LEXICON to various conceptual categories that have been defined in the DOMAIN MODEL ONTOLOGY, and establish semantic relationships between the terms. New terms and categories may be needed to express gradations of meaning for which there is no clear pre-existing vocabulary in the domain. The DOMAIN CONCEPT MODELS are used primarily in elaborating the DOMAIN FEATURE MODELS in the *Develop Feature Models* task.

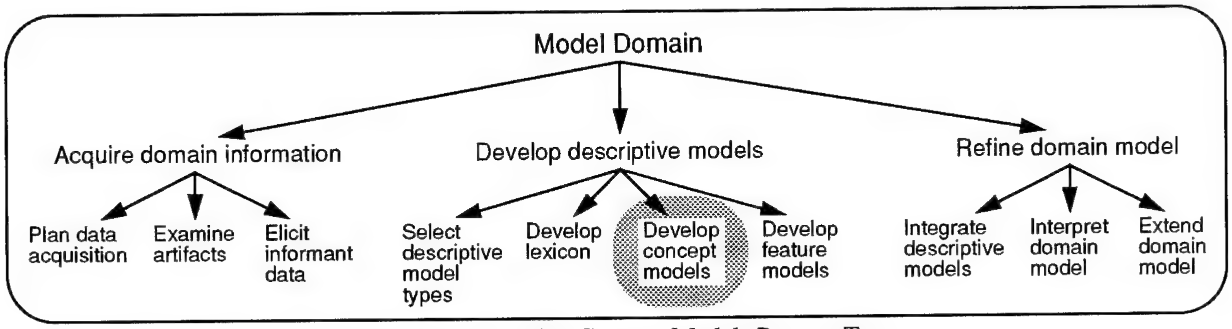


**Exhibit 43.** Develop Concept Models Process Tree

### Entrance Criteria

Comparative modeling of commonality and variability within domain data in the *Develop Concept Models* task can begin when:

- Modelers have completed the DESCRIPTIVE MODEL TYPES list, a component of the DOMAIN MODEL ONTOLOGY. Modelers need to know that the model they are developing is coordinated with the modeling effort as a whole. Otherwise models may have unanticipated overlaps, gaps, or inconsistencies in level of detail, representation, etc. The DOMAIN MODEL ONTOLOGY as a whole need not be complete. Detailed semantic relationships among models may be refined as part of modeling.
- The DOMAIN DOSSIER includes sufficient diversity of data to ensure that the DESCRIPTIVE MODELS developed are not skewed by a few special cases.
- The DOMAIN MODEL LEXICON is established, and procedures for migrating lexicon terms to the CONCEPT MODELS are in place. Models developed with uncontrolled vocabulary are difficult to correct afterward.
- Enough of the REPRESENTATIVE SYSTEMS SELECTION have been studied to provide a reasonable basis for ***taxonomic modeling***. Modeling should not begin on the basis of study of one system, nor be deferred until all systems are studied. A rough rule of thumb might be that, for a representative set of seven systems, three systems would constitute a good starting basis for modeling.
- Data acquisition should *not* be completed. If the start of modeling is delayed too long after the *Acquire Domain Information* sub-phase, comparative interpretations may filter into data gathering and be more difficult to sift out. In addition, some insights generated could be lost if data acquisition personnel are not involved with model development.

## Inputs

- <u>DOMAIN DOSSIER</u>. Source of the INFORMAL FEATURES, and other domain data resulting from the *Acquire Domain Data* sub-phase.
- <u>DOMAIN MODEL LEXICON</u>. Source of domain terminology that is migrated into the DOMAIN CONCEPT MODELS.

## Controls

- <u>DOMAIN MODEL ONTOLOGY</u>. Source that specifies which models to develop, and their semantic inter-connections.

## Activities

This task involves an interface to several supporting method areas, including Taxonomic Modeling and Conceptual Modeling. See Section 8.0 for more information about these supporting methods. Choices made in these areas will determine the actual representation formalisms used for the DOMAIN CONCEPT MODELS and the choice of the specific set of models to develop. These choices have probably already been encapsulated in the DOMAIN MODEL ONTOLOGY that is a control to this task. The following activity descriptions focus on the process aspects of the *Develop Concept Models* task. They should be read in conjunction with the relevant sub-sections of Section 8.0.

In addition, the activity descriptions that follow are not intended as a sequence of steps. They can be considered a repertoire of alternative strategies for deriving the DOMAIN CONCEPT MODELS from the various information sources available. These activities can be performed iteratively, in parallel, and/or selectively.

➤ <u>Identify analogous artifacts for comparison</u>

Working from the DOMAIN DOSSIER, identify sets of artifacts within different representative systems that appear to be ***analogues***, i.e., perform similar functions within their respective contexts. Working from the INFORMAL FEATURES describing these artifacts, look for terms and conceptual categories used in describing the semantics of analogous artifacts that can be related. Document these common conceptual categories, with their variable aspects, in the appropriate DOMAIN CONCEPT MODELS.

This activity marks a key transition point and hand-off between data acquisition and descriptive modeling. Only by identifying these analogous artifacts can clear exit criteria for data acquisition be established (i.e., that analogous artifacts are described to a consistent level of detail). But some initial description is needed in order for modelers to perceive the analogous functionality.

Establishing a structural framework for comparison across representative systems, and integrating the structural and taxonomic frameworks are key challenges in this activity. For systems with closely aligned structure (e.g., members of a systems family) this existing structure may be used as a basis for correlation. Where structures are diverse, modelers must construct a comparative framework distinct from the structure of the representative systems being compared, but that can be mapped to both. Alternatively, comparative modeling may have to proceed in a more bottom-up, fine-grained fashion, as described in the following activity.

➤ Map informal features into concepts

Starting directly from the INFORMAL FEATURES in the DOMAIN DOSSIER, decompose each feature into constituent conceptual elements as described above. The difference in the two activities is largely in the sequence performed. In this activity features are analyzed into concepts before looking for analogous artifacts in other systems.

➤ Analyze domain defining rules

Starting from the DOMAIN DEFINING RULES, a component of the INTENSIONAL DOMAIN DEFINITION, factor each rule into constituent conceptual elements. Look in particular for elements that appear in multiple rules. Assign each element to a model category within the DOMAIN MODEL ONTOLOGY, and add a representative element to the corresponding DOMAIN CONCEPT MODEL.

This strategy will result in links from defining rules to fragmentary concepts in models. As more defining rules are analyzed in this way, more concept elements are introduced into each respective model. Periodically in the process, switch from analysis to inspecting and re-organizing the concepts allocated to each model, so that there are semantic connections established among all elements in a model.

➤ Add semantic categories to lexicon terms

Working from the DOMAIN MODEL LEXICON, assign lexicon terms to the appropriate semantic categories from the DOMAIN MODEL ONTOLOGY. Add model elements and restructure the models as described in the previous activity.

➤ Scan data for each model

All the activities above share a common procedural approach of working from the data to the models. An alternative approach is to work from the DOMAIN MODEL ONTOLOGY. For each proposed model type, available data in the DOMAIN DOSSIER and DOMAIN MODEL LEXICON is scanned for conceptual elements appropriate to that model.

➤ Develop Comparative Models

Based on the differential information obtained from the various activities above, complete the models to express the range of commonality and variability observed in the data. These become the final DOMAIN CONCEPT MODELS. Introduces new terms where needed to allow for differentiation of concepts.

Caveat. The following bullets offer some suggestions for possible conceptual entities to be modeled. These are intended to be neither exhaustive nor prescriptive. It is assumed that each project will tailor this list to suit its own needs, and will discover other conceptual model types appropriate for its domain of focus. In fact, the conceptual model types suggested here should *not* be considered part of the core ODM method, but rather one example of a supporting method. They are included in the main section of this document to offer modelers a concrete starting point.

- Objects

  Objects are often the pivotal concept models in the domain, because they may represent the key abstraction around which domain operations are organized. The term is not intended to imply a strict object-oriented protocol, although object-oriented analysis techniques are one starting point for methods and representations. Some indicators of "object-hood" for a conceptual entity might be:

— correspondence with an agent or entity in the real world;

— a document or artifact in a system context;

— the ability to create multiple instances of the entity as a domain function;

— the ability to separately manipulate sub-components of the entity; and

— the entity's participation in larger ensembles as a sub-component.

> *Example.* In the outlining domain, different systems use terms to describe any outline heading, in contrast to those that have sub-headings beneath them in a given outline (e.g., "full" vs. "empty" heading). Outline documents themselves could be modeled as objects, with sub-structure. (See the first Guideline below for related issues.)

In modeling objects, both taxonomic and structural (or, for larger-grained objects, architectural) relations can be modeled. Taxonomic relationships can be defined between terms for objects when the terms represent different degrees of knowledge about the objects or when the specialization applies to a subset of the objects categorized by the parent.

- Operations

Operations, functions, or behavior concepts form a complementary model to the objects model. These concepts map most directly to the intuitive notion of features as user-visible functional capabilities.

> *Example.* Operations from the outlining domain include terms such as "promote" and "demote", "expand" and "collapse", "create new heading" etc.

Semantic relationships to model among domain operations can include the following:

— *Specialization.* For example, "collapse heading" vs. "collapse family" which acts recursively on all sub-headings within the outline.

— *Attributes of operations.* e.g., outliners operations have an associated scope of effect ranging over "character", "word", "current heading text", "whole outline", or "selected region".

— *Aggregation of compound operations.* e.g., "create aunt" could be modeled as a composite of "move to parent" and "create sibling".

Note that in these examples object terms are freely intermixed with operation terms. Operations will often not be fully factored out in raw domain data. There is an advantage to factoring them out separately in the DOMAIN CONCEPT MODELS.

- Relations

***Relations*** are second-order model entities based on objects and operations; i.e., object-to-object, object-to-operation, and operation-to-operation relations. For example, in the Outlining Domain, the term "headings" denotes a basic object in the outline structure. The terms "sister" or "sibling" are synonymous terms denoting a particular relation between headings.

- States/Conditions

Objects, operations, and relations provide a basic vocabulary that allows description of richer semantic constructs such as ***states*** or ***conditions***. These may refer to states of objects, or overall conditions in the system, the operational or use environment for the system.

This model should be distinguished from the more familiar use of states within representations such as state transition diagrams, which are designed to facilitate specification of the behavior of a single system under a set of input conditions. A taxonomic model of conditions or states classifies the significant variability in analogous condition and state descriptions across the representative systems. The model can in principle include sets of conditions not realizable by any single system.

As with objects, operations, and relations, conditions have linkages both to other models and semantic relationships to other conditions. For example, one condition could be said to specialize another if it implies an increase in the information available about the state of the entity being described.

- Error Semantics Model

  Error values often form an implicit taxonomy or classification scheme for unusual system conditions. Typically, system designers model such error conditions in a relatively flat error coding system with informal semantics. Some programming languages like Ada offer more semantic control over errors through exception-handling mechanisms, etc. Even in older languages, however, some taxonomic relations between error conditions can be deduced from error message text, conditional statements, error processing and user documentation.

  Of particular interest are errors that can be considered special cases of other error conditions, and understanding which errors take precedence when multiple error conditions occur simultaneously. These distinctions can lead to subtle variants in the behavior of components when lifted out of their original system-in-use context.

- Contextual Models

  Model common and variable characteristics of those contexts that have direct input on the domain artifacts of interest. Unlike states or conditions, which are more transitory and are used to specify the behavioral semantics of individual entities, contextual characteristics are more likely to affect the configuration of domain systems at a global level, i.e., will be characteristics of the context itself.

  An example of such a model is an environmental characteristics model. Such a model might characterize the range of variability in operational conditions that was factored into the design of a given system. Most system engineering models emphasize modeling the operational system itself. Assumptions about the environment in which a system will operate are sometimes only informally documented in requirements. In engineering for reuse, where multiple contexts of operation are a given, the potential variability across those environments must be considered. Identify aspects of requirements or system descriptions that encode information about the anticipated characteristics of the world outside the system. An example would be timing requirements expressed as a minimum or maximum interval to be handled between two external events driving the system. Such requirements indirectly state a set of expectations about the environment; these expectations can be modeled and compared with expectations embedded in the design of other exemplar systems.

## Workproducts

■ <u>DOMAIN CONCEPT MODELS</u>

DOMAIN CONCEPT MODELS are models of the commonality and variability for particular artifacts or objects across all the system contexts of all representatives in which this artifact or object occurs. An ***object*** is a real-world entity represented by data or by an artifact within a system context. DOMAIN CONCEPT MODELS can also describe processes (operations, human actions) or particular stakeholders within domain contexts. For example, in the Outlining Domain, the notion of a "heading" would occur within the concepts model. The DOMAIN CONCEPT MODELS show semantic relationships between domain terms. Exhibit 50 shows two example concept models in the outlining domain.

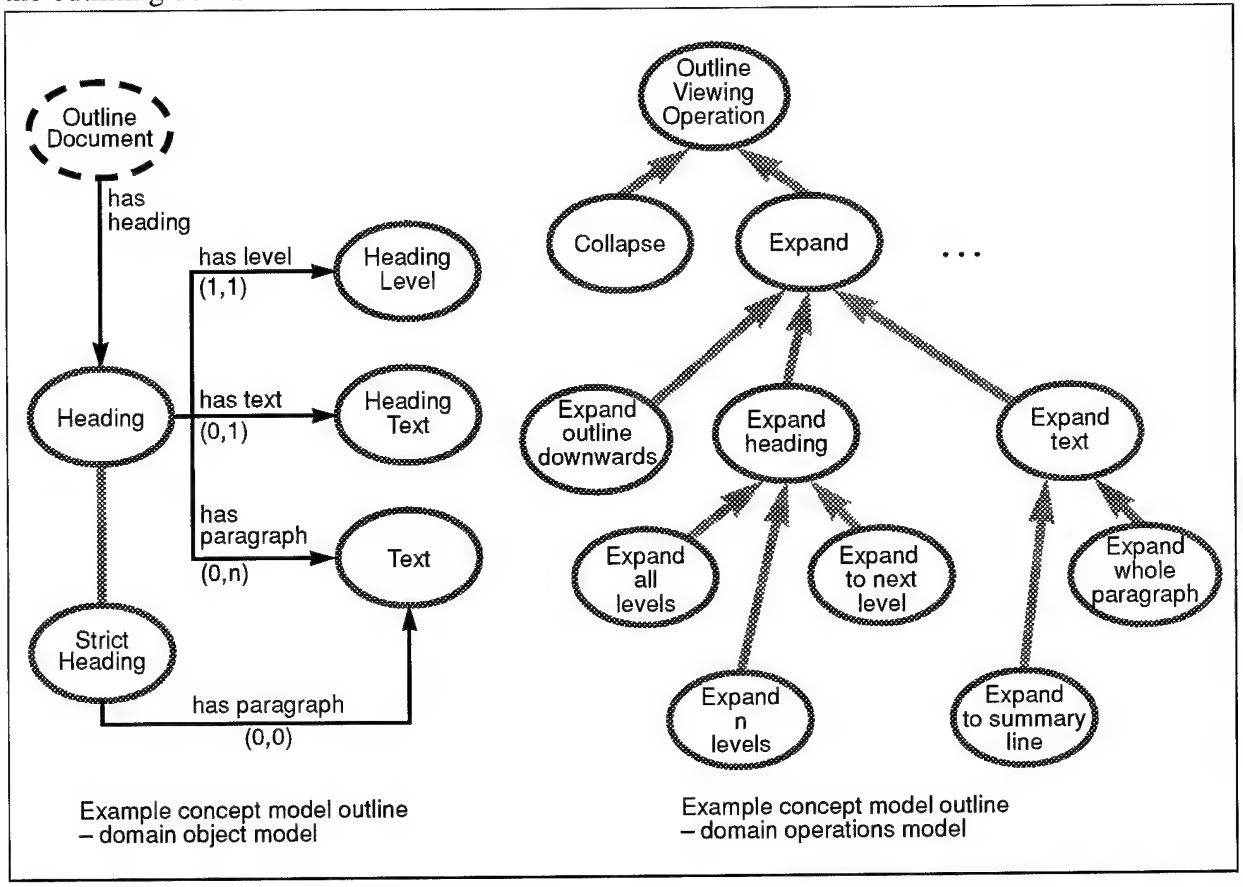


**Exhibit 44.** Example Concept Models

## Guidelines

- <u>Distinguish concept models from functional system models</u>. Concept models need to be carefully distinguished from functional models for a specific system, such as a data-flow diagram or a state transition diagram. These latter types of models are considered ***artifacts*** in ODM terms, because they describe the behavior of a single system, rather than commonality and variability across systems. Artifacts are examined (or, if necessary, generated) during the *Acquire Domain Information* sub-phase.

  Variability in system artifacts (e.g., diverse implementation languages used in analogous code modules of two representative systems) will generally be represented in features associated with a system-in-development rather than a system-in-operation or system-in-practice context.

An issue to track closely here is possible overlap between the notions of object and state, since one modeler might term an object in a different state as an object subclass. Other issues include possible confusion over the system context(s) in which an object exists. A patient and a patient record are closely related but not identical. A software component may be modeled as an artifact in the system-in-development context of the programmer, and take on a different semantics in the system-in-use context in which it executes. These complications are particular to the fact that conceptual modeling is being applied to the general domain of software development.

- Allow for changes in model types. Domain data can reveal new types of domains, not anticipated in the DOMAIN MODEL ONTOLOGY, but important for capturing the semantics of the domain. It may also prove best to reorganize the model types defined in the DOMAIN MODEL ONTOLOGY, e.g., splitting or consolidating models.

  Make sure to reflect these changes in updates to the DOMAIN MODEL ONTOLOGY and use it as a means of coordinating work with other modelers developing DOMAIN CONCEPT MODELS or performing activities within the *Acquire Domain Information* sub-phase.

- Consider hidden design assumptions in model partitioning Partitioning conceptual entities into categories such as objects vs. operations already embeds significant design decisions. Different representative systems may represent conceptual entities in diverse ways that suggest allocation to different models. Capturing the range of such representations for the exemplars studied will present significant challenges to the modeler's skill and will stress the representation formalisms being used as well.

  *Example*. In the Outlining domain, a system could implement an "outline presentation format" as an object, encapsulating a set of formatting decisions (heading fonts, etc.) that otherwise must be implemented as a series of individual operations.

- Use of object-oriented analysis techniques may prove useful here, with the following caveats.

  — ODM does not assume or require use of object-oriented techniques.

  — Object-oriented principles such as the encapsulation of methods with data should not be imposed upon the domain model if the exemplars studied do not exhibit this organization. Such restructuring would defeat the rationale behind descriptive modeling, unless done as reverse engineering to facilitate comparison of analogous artifacts and concepts.

  — Taxonomic relations (i.e., specialization with inheritance) on object categories have slightly different meaning in the domain modeling context than in the object-oriented context. Specialization among objects (as well as for other conceptual entities) has definitional import only, not operational import (e.g., code-sharing, etc.).

### Exit Criteria

The *Develop Concept Models* task is complete when:

- CONCEPTUAL MODELS are complete with respect to the model types specified in the DOMAIN MODEL ONTOLOGY.

- Key lexicon terms are allocated to conceptual categories.

- Constituent concepts in informal features have been modeled.

### Validation and Verification

- Formally validate individual models. Apply validation principles of the modeling formalism selected for each model.
- Validate consistency and completeness of separately developed models. Since the modeling task may be carried out by separate teams (either in sequence or in parallel) it is important to have a closure step that verifies the integrity of the models. This validation step can be compared to integration testing in conventional software development.
- Populate models with representative data from domain artifacts.
- Review models with selected domain informants.

## 6.2.4 Develop Feature Models

### Purpose

The primary purpose of feature modeling is to create a feature "language" rich enough to accommodate all variability in the domain as evidenced by the data culled from the domain's representative systems.

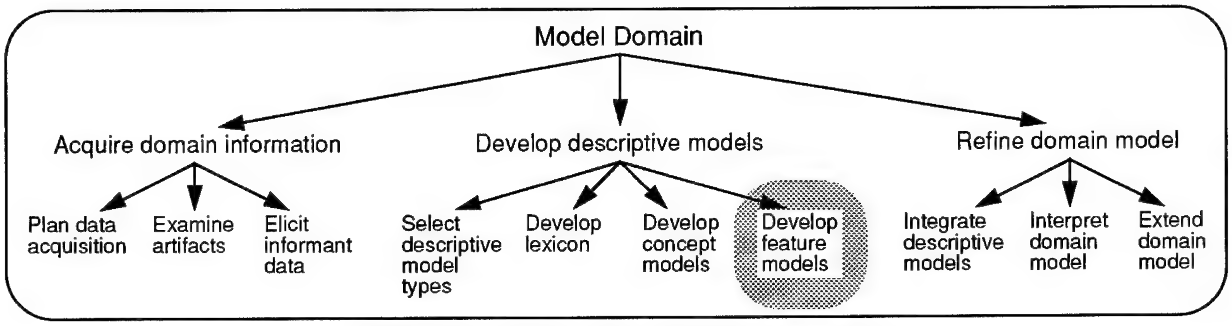


**Exhibit 45.** Develop Feature Models Process Tree

### Entrance Criteria

- The *Acquire Domain Information* sub-phase has produced some informal features for exemplar systems to be compared.
- The *Develop Lexicon* task has produced some key lexicon terms that can be used as the basis for the language used in feature statements.

The task can begin before the DOMAIN CONCEPT MODELS are complete. Sequencing the modeling of the conceptual and feature layer is part of the project tailoring. Trade-offs are discussed in the Description sub-section for the *Develop Descriptive Models* sub-phase.

### Inputs

- DOMAIN DOSSIER. INFORMAL FEATURES extracted from informant interviews, artifact descriptions and comparisons are a source for formal features. Features extracted from secondary domain artifacts (e.g., survey articles or interviews with a domain expert) may map most directly to formal features. Such sources of information will often include categorization of variation within the domain, without attribution to specific exemplar systems.

- DOMAIN CONCEPT MODELS. Source for features composed out of conceptual entities.
- DOMAIN DEFINITION. Defining features become roots of feature models. The DOMAIN INTERCONNECTION MODEL provides features of key interfaces.

## Controls

- DOMAIN MODEL ONTOLOGY. Provides an overall structure for types of models to be developed and their relationships. The DOMAIN MODEL ONTOLOGY may not enumerate each individual DOMAIN FEATURE MODEL in advance for this task, as these will emerge from the *Acquire Domain Information* sub-phase.

## Activities

Like the predecessor task, *Develop Concept Models*, this task involves an interface to several supporting method areas, including Taxonomic Modeling and Conceptual Modeling. These are described in Section 8.0. The following activity descriptions focus on the *process* aspects of the *Develop Feature Models* task. They should be read in conjunction with the relevant sub-sections of Section 8.0.

In addition, the activity descriptions that follow are not intended as a sequence of steps. They are a repertoire of alternative strategies for deriving DOMAIN FEATURE MODELS from the various information sources available. These activities can be performed iteratively, in parallel, and/or selectively.

➤ Extract features from DOMAIN DEFINING RULES

Every DEFINING RULE for the domain is converted into a domain feature, which becomes the root of a DOMAIN FEATURE MODEL that includes ***differentiating features*** for the domain. This activity can be performed in two distinct ways, depending on whether the modeling strategy being followed is concept-driven or feature-driven:

— In ***concept-driven feature extraction***, each DEFINING RULE is analyzed into constituent concepts. The feature is then built by linking the relevant entities in the DOMAIN CONCEPT MODELS. In this way the feature serves as an initial integration mechanism for the DOMAIN CONCEPT MODELS. Features can thus be defined that were not elicited as INFORMAL FEATURES during the *Acquire Domain Information* sub-phase, but are ***precedented*** within some representative system.

— In ***feature-driven feature extraction***, each DEFINING RULE is converted into canonic lexical form using terms from the DOMAIN MODEL LEXICON and then converted directly to a feature. The feature is then analyzed into its constituent concepts. This yields a sparser DOMAIN CONCEPTS MODEL that contains only concepts necessary for defining features.

The resulting model is ***adequate*** with respect to the REPRESENTATIVE SYSTEMS SELECTION (as represented at this point by the data in the DOMAIN DOSSIER) when there is a ***feature variant*** within the model that corresponds to the distinct characteristics of each representative system with respect to that feature category. When each key domain-specific term or phrase in an informal feature statement is linked to elements in the Domain Concepts Model, the feature statement is said to be a ***formal feature***.

➤ Extract formal from informal features

This activity is similar to the previous activity, except that INFORMAL FEATURES are used as the basis for creating formal features. Using this strategy, a larger set of features may be obtained, but there is no immediate principle for factoring these features into small individual models.

➤ Synthesize features from concepts

Extract features in a bottom-up fashion by examining DOMAIN CONCEPTS MODELS. This requires looking for characteristic patterns in concepts models that match a checklist of feature heuristics. For example, for every relationship defined between domain objects or sub-components of an aggregate structure, consider operations that probe for whether this relation holds between candidate objects.

➤ Develop the feature taxonomies

Based on the various elicitation strategies described above, develop a set of DOMAIN FEATURE MODELS in the taxonomic modeling system selected.

➤ Populate the models

***Populating*** a model involves adding descriptions of exemplar systems or artifacts as ***instances*** into the model in the appropriate categories. This is distinct from adding a new category or relationship to the model itself. Adding instances will initially necessitate adjustment to the model structure, and thus can be considered part of the natural evolution and validation cycle for each model. Evolving the domain model and modeling domain data are closely intertwined operations.Each model is complete when all representative systems have been characterized with respect to the main feature category in the model, and significant differences are captured in the model.

➤ Link features to domain contexts

Features are elicited from the standpoint of particular stakeholder perspectives and tasks in interacting with elements of systems within the domain. Features are contextually grounded to avoid confusion and ambiguity.

> *Example*. Consider features that might be associated with a requirements document. One feature might deal with characteristics of a system compliant with the requirements. Another feature might have to do with the nature of the requirements themselves as data objects, e.g., consistency, testability, etc. One feature is of interest to the requirements analyst who needs access to the characteristics of the requirements, while designers need access to the "content-oriented" features. The same artifact could have different features, appropriate to the interest, visibility, and tasks performed by various practitioners.

## Workproducts

■ DOMAIN FEATURE MODELS

Models of distinguishing characteristics of specific artifacts or representative systems as a whole. Each feature is an assertion or statement that holds for the associated system entity and indicates a significant differentiation from the standpoint of some domain practitioner. Exhibit 50 shows an example feature model in the Outlining domain.

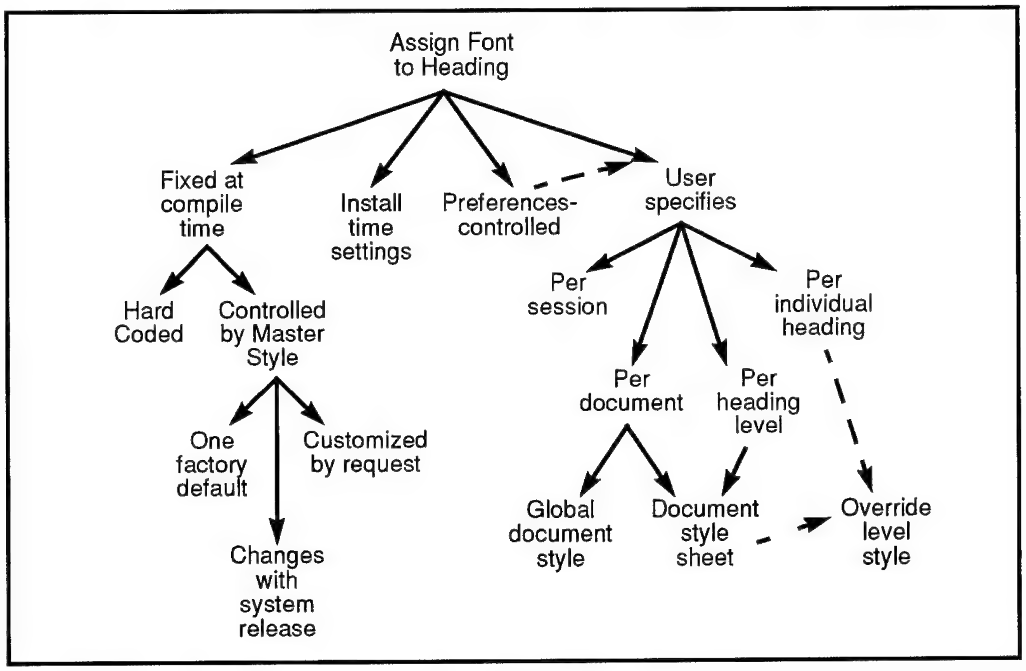


**Exhibit 46.** Example Feature Model

The feature models are developed using the same taxonomic modeling techniques used for the DOMAIN CONCEPT MODELS. Each feature is mapped to:

- associated concepts in the DOMAIN CONCEPTS MODEL.
- via concepts, to the DOMAIN MODEL LEXICON.
- via documented interests, to practitioners in the DOMAIN STAKEHOLDER MODEL.

## Guidelines

- Extract features from requirements artifacts. While technically these are merely one kind of artifact potentially studied during the *Acquire Domain Information* sub-phase, requirements have a special and closer relation to features because they are comparable to assertions made about the systems to which they apply. A feature of a given system can be interpreted as the capability of that system to respond to a given requirement (or set of requirements). Thus, there is a useful duality between requirements and features.
- Select complementary extraction techniques. Each technique has its advantages and disadvantages. Extracting features from the DOMAIN CONCEPT MODELS ensures that the formal semantics of features thus obtained follow very directly from the semantics in the concepts

models. On the other hand, it is possible to generate features of only academic interest in this way (i.e., features of no particular relevance to a stakeholder in the domain), while features important to key stakeholders may be missed. Extracting features from informant interviews will help identify relevant features but may require additional work to link descriptions with formal semantics of other models. We recommend that a minimum of one formal and one empirical technique be used in tandem to ensure that features have clear semantic relationships and practical relevance.

- Preserve but isolate innovative features. Processes throughout the ODM life cycle are intended to spur innovative thinking about domain functionality in a managed way, e.g., documenting analogy domains in the DOMAIN INTERCONNECTION MODEL. Features that seem intrinsic to the domain, but for which there is no precedent in the exemplar systems, can be handled in two ways:
  - The EXEMPLAR SYSTEMS SELECTION may be expanded to include new systems exhibiting the features at issue, which may in turn lead to selection of new representative systems for detailed modeling.
  - When domain modeling yields ideas for innovative (i.e., unprecedented) features, document them separately from precedented features. Innovative features can be modeled in the final task of the *Model Domain* phase, *Extend Domain Model.*

## Exit Criteria

The *Develop Feature Models* task is complete when:

- The models provides a language capable of expressing key differences across the REPRESENTATIVE SYSTEMS SELECTION. Key differences are those considered significant to domain stakeholders, as represented by the informants consulted during the *Acquire Domain Information* sub-phase.
- Each of the REPRESENTATIVE SYSTEMS SELECTION has been examined according to the DATA ACQUISITION PLAN. The resulting data has been accounted for in one or more of the DOMAIN FEATURE MODELS.
- Each defining feature of the INTENSIONAL DOMAIN DEFINITION has become the root of one of the DOMAIN FEATURE MODELS. The model contains differentiating features for that category that are sufficient to describe the variants observed within the REPRESENTATIVE SYSTEMS SELECTION.

The DESCRIPTIVE MODELS only need to note the commonality and variability across REPRESENTATIVE SYSTEMS SELECTION. They do not need to explain the rationale for the commonality and variability.

## Validation and Verification

- Every informal feature has been mapped to a corresponding formal feature in one of the DOMAIN FEATURE MODELS.
- For each separate DOMAIN FEATURE MODEL, each representative system has been characterized and the model is detailed enough to express significant differences between the systems with respect to that feature category.
- Each DOMAIN DEFINING FEATURE in the INTENSIONAL DOMAIN DEFINITION corresponds to a

high-level feature within a model.

- Each DOMAIN CONCEPT MODEL is associated with at least one DOMAIN FEATURE MODEL.

## 6.3 Refine Domain Model

### Purpose

The final output of the previous sub-phase, *Develop Descriptive Models*, is a set of individual DOMAIN FEATURE MODELS that, taken as a whole, provide a descriptive language for common and variant features within the REPRESENTATIVE SYSTEMS SELECTION. This set of purely DESCRIPTIVE MODELS is a necessary but not sufficient basis for selecting a set of features to implement in the *Engineer Asset Base* phase. The *Refine Domain Model* sub-phase of *Model Domain* serves as the transition from descriptive modeling in the *Develop Descriptive Models* sub-phase to prescriptive modeling in the *Engineer Asset Base* phase. The primary purpose of the *Refine Domain Model* sub-phase is to develop an explanatory model for the constraints and co-occurrences of features within domain exemplar systems. This explanatory model provides a basis for selecting what features are appropriate for a given market of customers. The explanatory model overlaps but need not coincide with the REPRESENTATIVE SYSTEM SELECTION. The explanatory model also provides a foundation for exploring innovative combinations and restrictions of features that improve the overall coherence of the resulting DOMAIN MODEL.

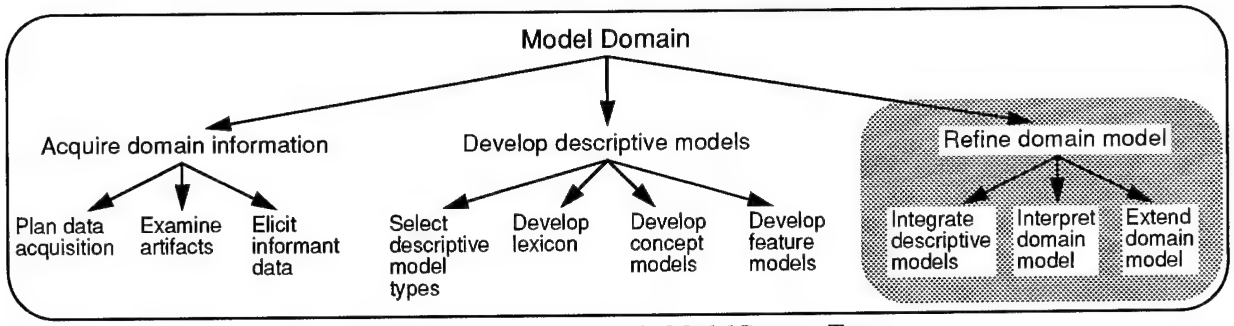


**Exhibit 47.** Refine Domain Model Process Tree

### Description

As shown in Exhibit 48, the *Refine Domain Model* sub-phase has three tasks:

- The *Integrate Descriptive Models* task unifies the separately developed DOMAIN FEATURE MODELS, along with the other DESCRIPTIVE MODELS, into a single INTEGRATED DOMAIN MODEL.
- The *Interpret Domain Model* task enhances the INTEGRATED DOMAIN MODEL with contextual information and rationale to explain why the representative systems differ in the observed ways.
- The *Extend Domain Model* task uses specific model transformation techniques to extend the purely descriptive model to a more coherent and orthogonal ***feature space***, and uses the contextual information from the *Interpret Domain Model* task to project potential contexts of use for various innovative feature combinations.

One key concept essential for understanding the tasks and workproducts of the *Refine Domain Model* sub-phase is the notion of ***feature binding sites***. Feature binding sites are points during the system life cycle at which feature variants may be selected. Depending on the domain context and the feature binding site, this selection may be done programmatically, by software, or by human decision-making. Different representative systems will make different choices with regard to

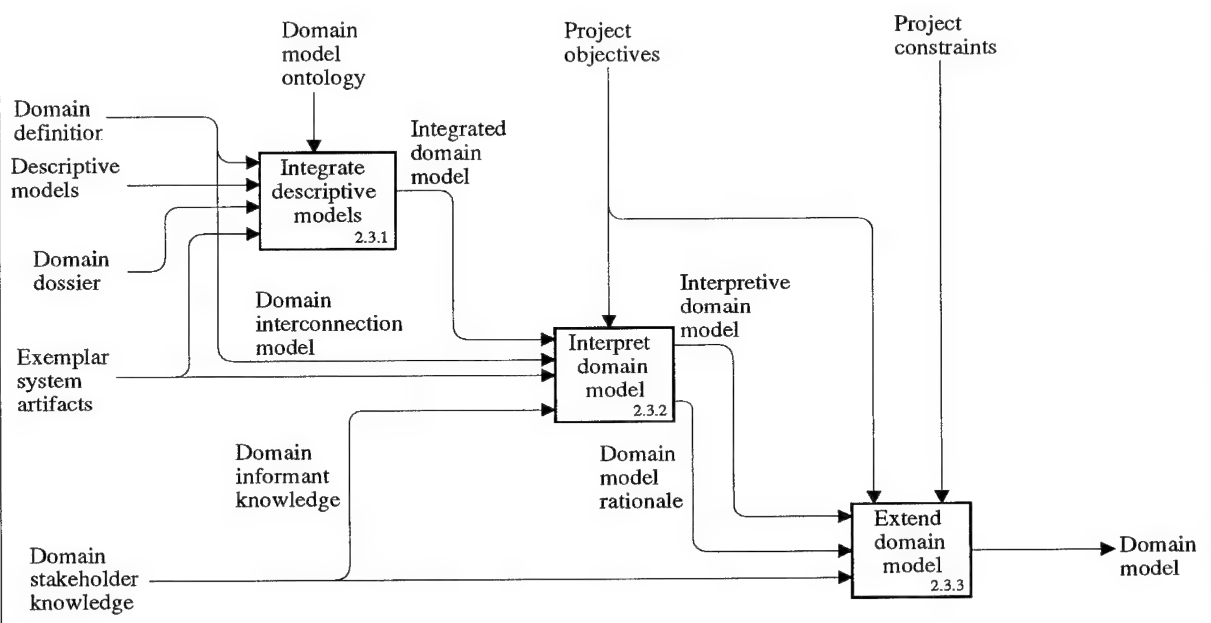

Domain model ontology
Project objectives
Project constraints
Domain definitior
Descriptive models
Domain dossier
Exemplar system artifacts
Integrate descriptive models 2.3.1
Integrated domain model
Domain interconnection model
Interpretive domain model
Interpret domain model 2.3.2
Domain informant knowledge
Domain model rationale
Domain stakeholder knowledge
Extend domain model 2.3.3
Domain model

**Exhibit 48.** Refine Domain Model $IDEF_0$ Diagram

where feature variants are bound. Some systems will have sites that do not occur elsewhere. Some will provide multiple options.

The ***system-in-use*** context in particular will often contain multiple feature binding sites. For example, user interface-oriented software will typically have a number of distinguished binding sites even within the operational environment of the system, e.g., system install time, session start-up time, tailorability through preferences menus, dynamic defaults, explicit commands, "next-operation-only" overrides.

> *Example.* In the Outlining domain, the feature "ability to assign a font to a given heading level" can be realized by operations performed at a number of different feature binding sites: e.g., by selecting a heading individually and changing the font, by setting a style for the entire outline, by having a system session default, by assigning the heading a style which can itself be reassigned, etc. In the case of a reassignable heading style, whether a change to a style affects headings already defined with the style or not is a more fine-grained binding-time issue. Modeling is required to equate the preferences file in one system with a differently named, but functionally equivalent "style format" in another system.

Each feature binding site has a set of ***practitioner*** roles who have ***visibility*** to features associated with that site. More generally, each site has a set of ***stakeholder*** roles for whom features are of ***interest***. Typically, the earliest practitioner in the domain-specific software life cycle who has visibility to a given feature also decides the variant that will be supported. This can be considered the ***deciding interest*** in a given feature.

> *Example.* The executable produced from a code component functions in the system-in-operation context. Various features can be associated with this context, such as system behavior under certain conditions and performance characteristics. These functional or operational

features will presumably be of interest to the end-user. These are the most straightforward kinds of features to model.

If some features of a component's functionality are configurable at the time the system was *installed*, this would still affect the overall functional profile, but the binding site comes at a different point in the life cycle. In this case, the system installer role would have visibility to the features at the "install-time" binding site, whereas end-users would still have interest in the feature variants selected.

The extent to which software functionality is embedded in hardware components will affect where configurability options are available. Embedded software or firmware in a printer or other peripheral product may require specialized personnel such as field technicians or site system managers, each of whom may have access to particular features of interest and the capability to shape the performance of the system.

Role of feature binding sites in the *Refine Domain Model* sub-phase. In the *Develop Feature Models* task within the *Develop Descriptive Models* sub-phase, modelers first identify specific features bound at different sites, typically in the system-in-operation context. Since the binding sites for features may vary across systems, or even within a given system, the comparative data needed to identify them only becomes systematically available during the *Integrate Feature Models* task, when the separately developed DOMAIN FEATURE MODELS are linked into a single INTEGRATED DOMAIN MODEL.

Later, during *Interpret Domain Model*, links are established between feature variants that represent the same "logical" feature projected across different binding sites in different domain contexts. The litmus test for this "logical" association of features required is really the viewpoint of domain practitioners. Establishing these links requires both the INTEGRATED DOMAIN MODEL and additional DOMAIN INFORMANT KNOWLEDGE that allows correct interpretation of the feature semantics.

Finally, in *Extend Domain Model*, new binding sites for precedented features can be explored by shifting features to new domain contexts (e.g., compile-time to run-time). During the *Scope Asset Base* sub-phase of *Engineer Asset Base*, these various precedented feature variants and potential innovations are evaluated for usability and feasibility, and a subset of these is selected for the intended scope of the asset base to be developed.

## Sequencing

The *Integrate Descriptive Models* task should be complete before beginning the *Interpret Domain Model* and *Extend Domain Model* tasks. The *Interpret Domain Model* and *Extend Domain Model* tasks each have the potential for an explosion of scope without the grounding of the fully INTEGRATED DOMAIN MODEL as a starting point.

Some iteration between the *Interpret Domain Model* and *Extend Domain Model* tasks is natural. Innovative ideas for features need to be validated by reinterpreting existing data; conversely, eliciting explanatory rationale from informants may require some hypothesizing of alternatives. The tasks can be done in parallel; however, managing this concurrency will require extra discipline on the part of the modeling team.

## 6.3.1 Integrate Descriptive Models

### Purpose

The domain modeling phase represents a process that can be implemented with a variety of strategies leading to separate descriptive models. While ODM does not prescribe a particular strategy, the method does acknowledge that in practical settings, some divisions of this kind will be necessary. Building separate models, then integrating them into a unified model, seems to be an essential dynamic of the overall process. In the *Integrate Descriptive Models* sub-phase, the DOMAIN MODEL LEXICON, DOMAIN CONCEPT MODELS and DOMAIN FEATURE MODELS are merged into one INTEGRATED DOMAIN MODEL.

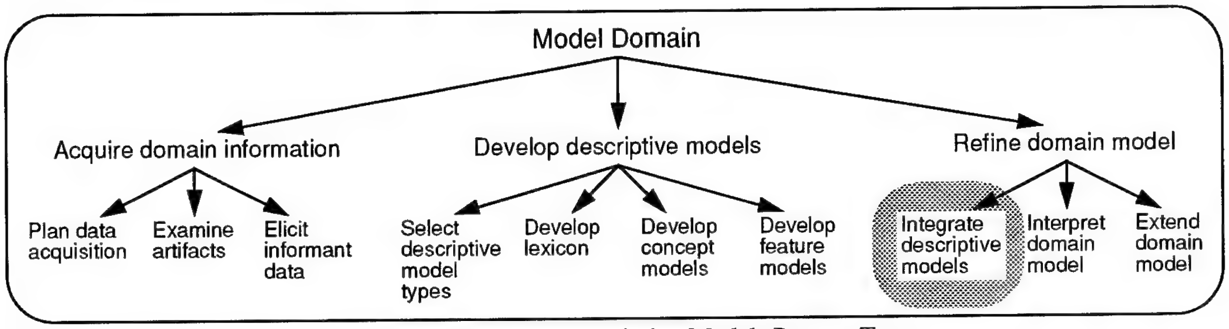


**Exhibit 49.** Integrate Descriptive Models Process Tree

While analogous is a general way to the notion of unit testing and integration testing in building a single system, the analogy breaks down because a domain asset base is not built in response to a single set of requirements. The separate viewpoints, alternative orders of seeing and comparing data are important for robust cognitive results of the process.

This validation activity takes place at the beginning of the refinement stage because integration issues that are uncovered at this stage generally will require interpretation to be resolved, and may thereafter lead to useful extensions of the domain model. In this sense the entire refinement process can be seen as having a validation function.

The *Develop Descriptive Models* sub-phase serves as a kind of overall integration phase with respect to domain data; comparative modeling integrates separate data into a comprehensive model of commonality and variability.

Some of the benefits of this sub-phase are as follows:

- Support interpretation of model data. It is only after integration of separate aspects of the model that the necessary insights can be obtained to uncover hidden contextual information and rationale. We often don't ask why a system has a given feature until we have seen an example of a system with a variant feature.
- Provide a single model for future iteration. After completion of domain modeling, subsequent evolution steps can iterate back to the INTEGRATED DOMAIN MODEL, rather than back to individual DOMAIN FEATURE MODELS. These thus take on more of the nature of transitional workproduct, which reflect artifacts of team strategy, representation limitations, the order in which data was examined, modeler bias, etc. Evolutionary extensions may take the form of new data, new individual feature variants, or entire new ranges of features.

## Entrance Criteria

You are ready to *Integrate Descriptive Models* when:

- A set of individual DOMAIN FEATURE MODELS have been completed and all required validation for individual models has been performed. In this regard, this task is analogous to integration testing following unit testing in conventional software development. Integration activities can be performed before all the individual DOMAIN FEATURE MODELS are complete. However, it is best not to think of this task as receiving a steady "pipeline" of updates to DESCRIPTIVE MODELS. Some early integration activities can be done to validate the planned integration mechanisms, but the major part of the task should be performed when the *Develop Descriptive Models* sub-phase is complete. Otherwise a lot of repeated effort will be incurred.
- The *Acquire Domain Information* sub-phase is nearly complete for at least *one* of the REPRESENTATIVE SYSTEMS SELECTION. Part of this task involves verifying the adequacy of the INTEGRATED FEATURE MODEL to fully characterize these systems in terms of domain features.

## Inputs

- DESCRIPTIVE MODELS. The separate models to be integrated.
- DOMAIN DOSSIER. Information and characterization of the REPRESENTATIVE SYSTEMS SELECTION, used to derive the REPRESENTATIVE SYSTEMS FEATURE PROFILE.
- DOMAIN DEFINITION. The INTENSIONAL DOMAIN DEFINITION and DOMAIN INTERCONNECTION MODEL are used to establish the boundaries and interfaces to the INTEGRATED DOMAIN MODEL.
- EXEMPLAR SYSTEM ARTIFACTS. For validating the model against an exemplar outside the representative set

## Controls

- DOMAIN MODEL ONTOLOGY. Used to structure the semantic relationships of the separate models into a single integrated model.

## Activities

➤ Integrate features

Using the DOMAIN MODEL ONTOLOGY as a structuring framework, integrate the separate DOMAIN FEATURE MODELS. Eliminate redundancies across models, resolve naming inconsistencies, identify apparent gaps in the models, and establish the required linkage so that a given artifact, or a given system in the domain, could be characterized by a single, minimally overlapping set of features. Exhibit 50 shows an example of two feature models to be integrated in the Outlining domain.

➤ Integrate contextual data

To the extent that contextual information has been modeled in the DOMAIN CONCEPT MODELS, integrate these models with the features. This activity will be completed more comprehensively within the subsequent *Interpret Domain Model* task.

➤ Identify Feature Binding Sites

For each ***domain context***, identify the set of distinct ***feature binding sites***: points during the system life cycle at which feature variants may be selected. Refer to the Description sub-section of Section 6.3 for more information on feature binding sites. Identify feature binding sites by the following procedure:

1) Compare the different feature variants occurring in the separate DOMAIN FEATURE MODELS that are associated with this domain context.

2) Cluster variants associated with the same practitioner roles (e.g., requirements analyst, designer, end-user); and/or compare practitioners' work scenarios and look for common decision points for different feature variants (e.g., several variants are determined when a given configuration file is edited).

3) Distinguish the different sites if the activity or mechanism differs, if the decision points happen at distinct times, or if different practitioners have responsibility. This is the rationale for using the more general term "binding site" rather than the more conventional "binding time" which draws a clearer analogy to "variable binding time" in programming. In the domain engineering context, there is not always a single temporal line along which the various sites can be positioned. The binding site encompasses the elements of time in workflow, time in

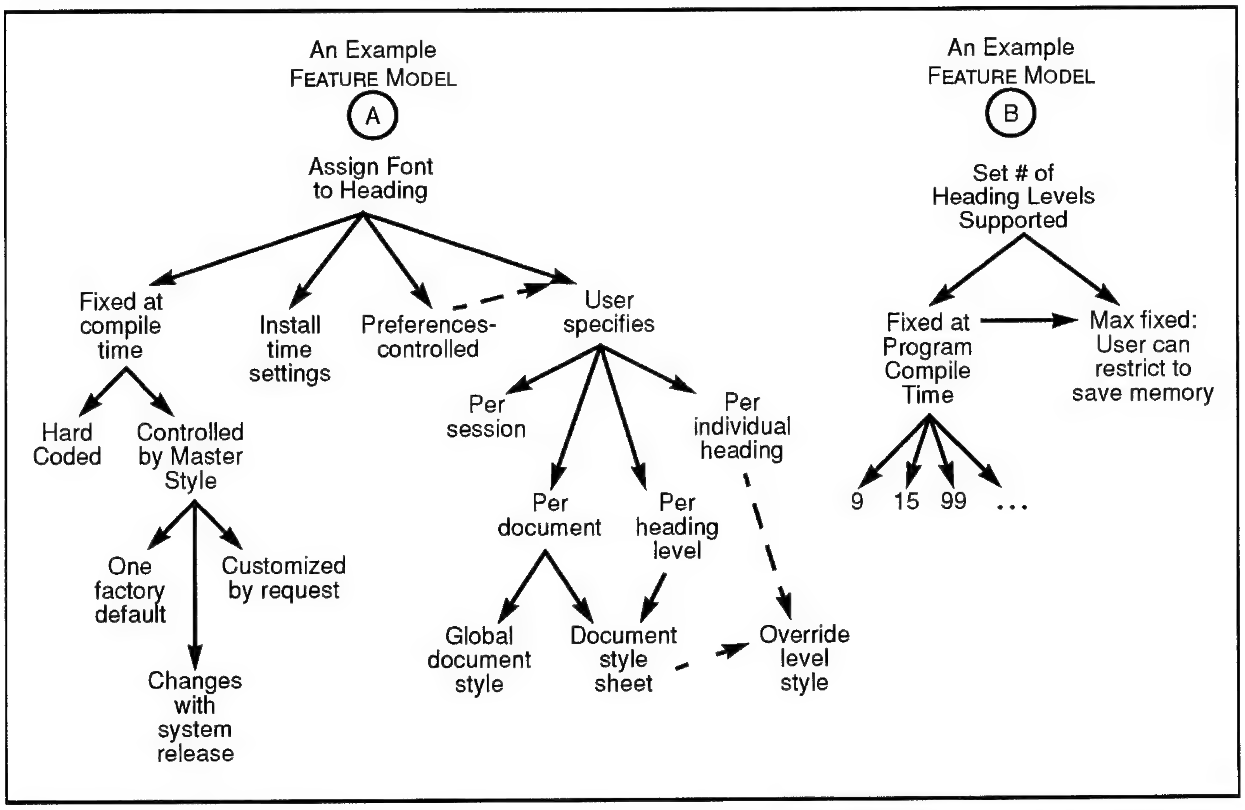


**Exhibit 50.** Examples of Feature Models to Integrate

program execution, and different practitioners in different contexts.

4) Document the binding sites identified for each context, together with the associated practitioner roles, in the FEATURE BINDING SITES MAP component of the INTEGRATED DOMAIN MODEL. Use names for the binding sites that are familiar to domain practitioners. Some names may have been previously included in the DOMAIN MODEL LEXICON. If so, the DOMAIN FEATURE MODELS may already be structured in terms of the feature binding sites.

5) Re-organize individual DOMAIN FEATURE MODELS so that variants are differentiated where appropriate by reference to the associated feature binding site for the variant. Depending on the degree of formality in the domain modeling representation, the FEATURE BINDING SITES MAP may be converted into an explicit "FEATURE BINDING SITES MODEL" that is incorporated into the final INTEGRATED DOMAIN MODEL, but this is not required.

In the simple example shown in Exhibit 50, each separately developed feature model embeds features that span the compile-time and run-time feature binding times (sites, once placed in a specific context). The next step would be to re-organize the models so that this information was factored into a common, integrated model.

➤ Characterize representatives with integrated models

Using the completed INTEGRATED DOMAIN MODEL and the information from the DOMAIN DOSSIER, characterize a reasonable subset of the REPRESENTATIVE SYSTEMS SELECTION using the feature model. Characterize in terms of individual functional features, global, system-wide features and contextual information. Record the results in the REPRESENTATIVE SYSTEMS FEATURE PROFILE.

## Workproducts

■ INTEGRATED DOMAIN MODEL

- FEATURE BINDING SITES MAP. Integrates stakeholders with interests in features from the DOMAIN STAKEHOLDERS MODEL, domain contexts, and associated roles of practitioners with visibility/access to features, from the DOMAIN CONTEXTS MODEL. The FEATURE BINDING SITES MAP refines these relationships down to the next level of granularity, to the individual feature binding sites within each domain context and their relationships to practitioner roles.

- INTEGRATED FEATURE MODEL. A linked version of the separate DOMAIN FEATURE MODELS with redundancies and inconsistencies removed.

- REPRESENTATIVE SYSTEMS FEATURE PROFILE. A classification of each representative system in the REPRESENTATIVE SYSTEMS SELECTION in terms of the INTEGRATED FEATURE MODEL.

## Guidelines

- Do incremental integration. Although the process model may imply otherwise, integration can and should be done throughout the modeling process. Use the DOMAIN MODEL ONTOLOGY and the DOMAIN MODEL LEXICON to control divergence of models, and frequent team reviews to make sure that the modeling efforts are applying consistent conventions.

## Exit Criteria

The *Integrate Descriptive Models* task is complete when:

- All workproducts of the *Define Domain* sub-phase of *Plan Domain Engineering* are complete. These have been integrated with the DESCRIPTIVE MODELS and conflicts have been resolved.
- All workproducts of the *Develop Descriptive Models* sub-phase of *Model Domain* have been integrated and conflicts resolved. These include the DOMAIN MODEL LEXICON as well as the DESCRIPTIVE MODELS.
- The INTEGRATED DOMAIN MODEL provides an adequate basis for answering the question "What specific features differentiate the REPRESENTATIVE SYSTEMS SELECTION?"

### Validation and Verification

This task has a validation and verification purpose as a whole.

- Validate completeness of the feature model. Is the INTEGRATED FEATURE MODEL an adequate basis for distinguishing representative applications systems as a whole? Have some important features been left out? Does the constitute a rich enough language for distinguishing profiles of systems in the domain?
- Characterize a non-representative from the exemplar set with the integrated feature model. This is the jumping-in point for evolving the feature model on the basis of incremental asset base feedback.

## 6.3.2 Interpret Domain Model

### Purpose

The INTEGRATED FEATURE MODEL for the domain represents the required range of variability for the representative set of systems. Some rationale for the features of particular artifacts or representative systems may have been documented in the *Acquire Domain Information* sub-phase, but before the model of features was complete there was no systematic way of deriving rationale for the variability. The primary purpose of *Interpret Domain Model* is to establish contextual rationale for *why* particular systems have particular features, and why the features differ across the representative set.

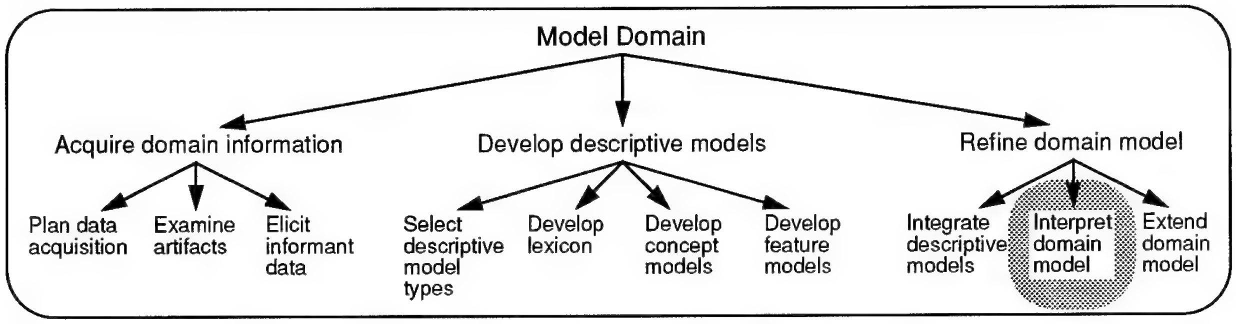


**Exhibit 51.** Interpret Domain Model Process Tree

Within the *Model Domain* phase, this rationale will provide a filter on the various techniques used to formally extend the domain model into its final form (performed in the following process, *Extend Domain Model*). Rationale for various features in the domain model will later be used in the *Engineer Asset Base* phase to identify potential customer interest in particular feature combinations. Finally, this contextual information will be a key input to the *Implement Infrastructure* process, where it will be incorporated into mechanisms to support asset utilizers in searching the asset base.

## Entrance Criteria

The *Interpret Domain Model* task can begin when:

- The *Integrate Feature Models* task is complete. A comprehensive and accurate picture of the significant features in the domain provides the basis for eliciting relevant rationale and contextual information.
- Feature profiles are completed for some subset of the REPRESENTATIVE SYSTEMS SELECTION. These provide a basis for interpretation of variability in domain data. Interpretation may require elicitation of further information (e.g., analogous to further experiments to validate a scientific hypothesis) but should *not* involve iterations to verify initial data.

## Inputs

- INTEGRATED DOMAIN MODEL. The primary input, which provides the features for interpretation. The REPRESENTATIVE SYSTEM FEATURE PROFILE provides profiles of representative systems in terms of the integrated set of features. The DOMAIN DOSSIER is also considered part of this input.
- DOMAIN INTERCONNECTION MODEL. Provides analogy domain relationships as one source for interpretation of feature configurations. Genealogical information at the domain level may also be significant.
- EXEMPLAR SYSTEM ARTIFACTS. Required for new data acquisition activities undertaken to validate interpretations.
- DOMAIN INFORMANT KNOWLEDGE. Newly elicited knowledge about historical and genealogical relationships among REPRESENTATION SYSTEMS. needed to distinguish historical rationale for patterns of feature occurrences within context.

## Controls

- PROJECT OBJECTIVES. Focuses interpretation activities on the needs of potential asset implementors and asset utilizers.

## Activities

➤ Map feature co-occurrences

Working from the REPRESENTATIVE SYSTEM FEATURE PROFILE within the INTEGRATED DESCRIPTIVE MODEL, look for significant patterns of co-occurrence across systems. These include ***feature clusters*** that co-occur in several systems, as well as features that seem to rarely co-occur. The larger the set of representative data, the larger the cluster, and the larger the set of co-occurrences, the more significant the cluster is from the standpoint of domain "evidence". Document these by annotating the REPRESENTATIVE SYSTEM FEATURE PROFILE.

➤ Form hypotheses and questions

Based on observed feature clusters and other patterns, form hypotheses about the rationale for the co-occurrence patterns. Use documented rationale from the DOMAIN DOSSIER as a starting point. Also use any

Consider the historical relationships between representative systems. A primary heuristic to apply here is the following:

> The closer the historical or contextual closeness of two representative systems, the more interesting is the variability in the feature profiles. Conversely, for systems that are historically or contextually diverse, commonality in feature profiles is of particular interest.

Document the results in a list of hypotheses and questions for investigation that would validate the hypotheses.

➤ Elicit supporting data

Use the questions as basis for further elicitation, artifact analysis, and observation. Acquire information about features using the same techniques and supporting methods as were employed in the *Acquire Domain Information* sub-phase of *Model Domain.*

It is often difficult to recreate contextual information by working strictly from typical static sources (e.g., documents, code, etc.). Study of system users' practice may reveal the rationale for the functionality of certain features, or may reveal that some features are part of the ***system-in-practice*** contextual layer, e.g., the functionality is performed through system workarounds, routine sequences of primitive system operations, informally reused data, etc.

> *Example.* In the Outlining domain, outlining programs that evolved from an "idea-processor" perspective differ in fundamental ways from programs that treated outlining as an extension of document processing. While both categories are considered outliners, the set of operations supported and even the underlying data structures differ considerably.

➤ Model constraints

Constraints, in the context of the DOMAIN FEATURE MODELS, might be best thought of as "negative features". Constraints are valuable in differentiating features that are intentionally rather than accidentally excluded from the domain range of variability, or cases where a supported feature in one context is specifically required to be absent in another context. Constraints can simplify other models, by selectively filtering excluded instances, thus allowing the taxonomic model to define a superset of the instances intended for coverage. They can help control the potential combinatorial explosion of variation in domain modeling.

> *Example.* In the Outlining domain, one system enforces constraints that disallow creation of outlines in "improper style" (i.e., a heading of level 3 immediately following a heading of level 1). Other programs allow such configurations. The difference is *not* merely one of greater versus lesser functionality. If the intended use of the tool is as a strict outline manager or idea processor, the more restrictive version of the tool may in fact be more useful. The less constrained variant is patterned more after the notion of styles in a document format, and is more typical in applications where outlining is embedded within broader text-editing capabilities.

The degree to which constraint modeling can be directly integrated with concept and feature models depends on the sophistication of the modeling representations used, the specific constraint mechanisms they support, and the presence of other representation capabilities such as multiple inheritance.

➤ Cluster features into ensembles

As the final activity in the *Interpret Domain Model* task, synthesize the contextual and interpretive information derived by documenting large-scale ***feature ensembles*** that correspond to identifiable attributes of contexts of application. This step is a transition to the next task, *Extend Domain Model*, where innovative combinations of features will be explored.

> *Example.* Features in the domain of interactive text editors could be partitioned according to those that support a moded vs. a modeless style of operation. While in principle "modedness" itself could be considered a feature, its semantic repercussions, interactions and impact on other features is so pervasive as to warrant its representation as a related ensemble of feature variant choices.

## Workproducts

■ INTERPRETIVE DOMAIN MODEL

INTEGRATED DOMAIN MODEL extended with:

- — Features clusters with strong rationale for co-occurrence.
- — Some feature occurrences documented as historical "vestiges" rather than strongly correlated with contextual profiles.
- — Feature constraints, which effectively reduce the possible feature space to exclude combinations interpreted as out of scope or semantically meaningless.
- — Contextual model describing attributes of domain practitioners or surrounding system context that are relevant to the explanatory model of features derived in this task.

■ DOMAIN MODEL RATIONALE

Documents process information, rationale, decision histories, trade-offs, and issues surrounding elements in the INTEGRATED DOMAIN MODEL. In the *Acquire Domain Information* sub-phase, rationale may be gathered about specific representative systems or artifacts. In *Interpret Domain Model*, however, the intent is to develop rationale explaining the commonality and variability in the DESCRIPTIVE MODELS. Usually rationale is specified at the level of the INTEGRATED DOMAIN MODEL, although some information could also be recorded about finer-grained conceptual entities such as objects and operations (e.g., when might a particular operation be useful, what are its performance penalties).

## Guidelines

- Be alert for functionality withheld from end-users for definite and deliberate reasons. It can be as important to explain the absence as well as the presence of a feature in a given context. Document such discoveries in updates to the DOMAIN MODEL RATIONALE and/or the INTERPRETIVE DOMAIN MODEL.

*Example.* In some military applications where normally both read and write capabilities might be expected, write capabilities are intentionally factored out of implementations to avoid malicious changing of orders or for other security reasons. A system providing both read and write access would not be acceptable in this context.

*Example.* An interesting example of the use of an ***analogy domain*** relationship to elicit hidden contextual information comes from the Army/Unisys STARS Demonstration Project. Discussing the domain of Emitter Location Processing and Analysis (ELPA) in an information-gathering meeting, an analogy was made to car radio receivers. The analogy arose because, like drivers with their car tuners, ELPA operators can initiate operations to "seek" or "scan" various frequency ranges. By working backwards from the analogy, the question arose: did operators have functionality analogous to the pre-set station buttons on a car radio? The answer was no and the rationale was "... to prevent system operators from using the receivers as radios for entertainment."

### Exit Criteria

The *Interpret Domain Model* task is complete when:

- The INTERPRETIVE DOMAIN MODEL can answer questions about the rationale for observed commonality and variability in the REPRESENTATIVE SYSTEMS SELECTION. The model adequately answers the question "Why are the systems different in the observed respects?"
- For a given descriptive feature or feature set, it is possible, using the model, to generate a ***contextual profile*** that would be a good match for the features.
- Conversely, for a given contextual profile, key features of interest can be derived.

### Validation and Verification

- Validate data with informants in interaction. This is a good point to consider bringing informants into contact with each other as part of the validation process. Generating rationale after the fact can give way to some speculation; let informants provide a cross-check on their own data. This is harder to do earlier in the modeling phase.
- For each significant point of variability in the model, can you answer: Why are these systems different (e.g., in the way they handle feature X)?

## 6.3.3 Extend Domain Model

### Purpose

This process provides an opportunity for innovative design and the discovery of novel opportunities, through combinations of features that have all been proved individually to be attainable. It can help reveal unexpected commonality across systems and even across domains. It also lays the groundwork for a layered design approach that identifies core and peripheral sets of features for system versions within the domain. This layered approach may first be suggested in *Interpret Domain Model* if there are several distinct overall semantic interpretations for structures and operations in the domain. It also may carry over into *Architect Asset Base*, where ability to optimize variants with the right mix of features may be a condition of getting assets used.

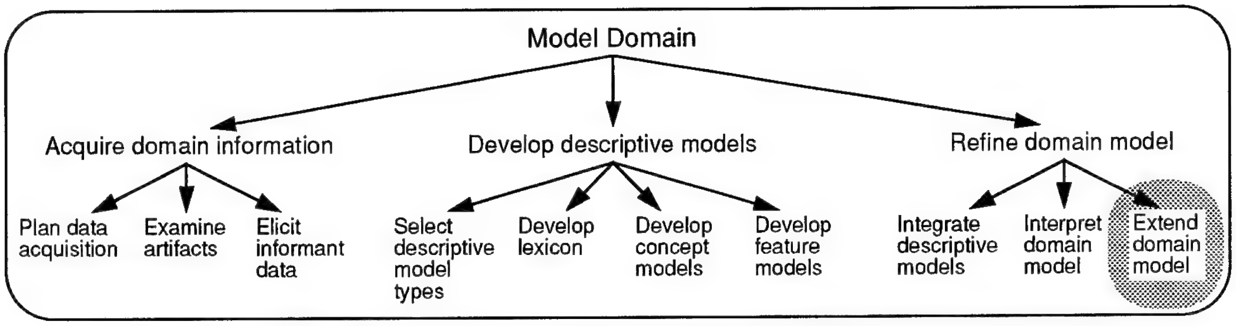


**Exhibit 52.** Extend Domain Model Process Tree

The DOMAIN MODEL does not represent commitments on the part of modelers to implement all features and feature combinations expressed. Determining the intended scope of feature coverage for the domain asset base is done in the *Scope Asset Base* sub-phase of *Asset Base Engineering*.

A key risk in this process involves a potential combinatorial explosion of variants within the model. Use of a modeling formalism with capabilities such as multiple inheritance, representation of constraints, and selection of aggregate configurations, can help to mitigate this risk. Aspects of the ODM approach that provide some control over this complexity of feature combinations include:

- Extra care taken in defining domain boundaries.
- Isolating feature subsets of the problem domain as subdomains treated as "black box" elements within the feature model.
- Emphasizing relatively small domains, with manageable numbers of features but richly modeled semantic relationships.

## Entrance Criteria

Do not initiate the *Extend Domain Model* task until:

- The INTERPRETIVE DOMAIN MODEL is complete. This documents constraints in domain feature interactions that help reduce the number of combinations explored for the EXTENDED DOMAIN MODEL.
- The previous tasks in the process have created a shared intuition among the modelers of what features belong and do not belong in the domain. Without this shared understanding, robust enough to be applied to fine-grained feature descriptions, the *Extend Domain Model* task may seem an empty exercise, or worse, may seriously compromise the integrity of the modeling process that has been so carefully maintained in previous activities.

Ideas for innovations in domain functionality will occur throughout the modeling process. These can be captured informally as they arise (e.g., in a "things we would like to see" list) and used as an source of fruitful ideas, and as input into the formal *Extend Domain Model* task. The *Extend Domain Model* task should not be initiated until the above criteria are met.

## Inputs

- INTERPRETIVE DOMAIN MODEL. The model of descriptive features and their contextual interpretation, used to derive innovative features.

- DOMAIN MODEL RATIONALE. Rationale for the variability in the domain. Used to derive potential contexts of application for innovative features.
- DOMAIN STAKEHOLDER KNOWLEDGE. Must now have access to full stakeholder knowledge, not just informants

**Controls**

- PROJECT OBJECTIVES. Provides focus for selection of particular transformation techniques or areas of emphasis in the task, e.g., optimizing the DOMAIN MODEL for ease of understanding where training of new personnel in the domain is an objective, optimizing for minimal feature-set configurations in a domain where performance constraints are a significant factor.
- PROJECT CONSTRAINTS. Provides a filter both on level of effort for activities and on the degree to which the model can be extended.

**Activities**

Rather than offering a systematic linear process for innovation modeling, the approach taken in ODM is to describe a repertoire of specific techniques for model innovation that can be applied opportunistically, based on perceived characteristics of the descriptive models. These techniques are described below.

➤ Close feature space (orthogonalization)

When features are semantically linked to concepts such as objects, operations and relations, it is possible to derive novel variants by shifting an existing variant with respect to a related concept. This technique can yield extremely fine-grained variants within the feature model. It generally relies on the separation of concepts in ways that may conflict with some established software engineering techniques For example, where an object-oriented approach would advocate binding operations (as methods) to object definitions, maintaining separate object and operation models in the DOMAIN CONCEPT MODELS layer allows for orthogonal combinations leading from either the objects or the operations.

➤ Feature Restriction

During *Interpret Domain Model*, regularly co-occurring feature clusters were examined to determine whether the co-occurrence is logically or merely historically based. During *Extend Domain Model*, identified clusters can be restricted to elicit new, leaner feature combinations and profiles. Restriction may include suppression of one or more features from a cluster, or replacement of a feature with a specialized variant.

➤ Feature Binding Time Variants

Feature variants are associated with particular binding sites with respect to the various contexts of the domain-specific system life cycle. Methodically shifting features to other binding sites within the domain-specific system life cycle can suggest previously unanticipated potentially useful new capabilities. For example, it may be possible to link in various algorithms with differing performance characteristics, when a system is configured for a site. Run-time algorithm selection by system users could also be considered. Developers often provide ancillary functions for their own environment that would be useful to end-users as well, e.g., test and simulation capabilities.

The innovation need not always consist of transforming a static to a run-time feature, however. For example, the development of "spreadsheet compilers" came historically after conventional spreadsheet programs. Spreadsheet compilers split the schema definition and data entry tasks into separate, independent components of the system, using a generative step after schema definition to produce a data entry program optimized (and hard-coded) for a particular spreadsheet schema. Data entry personnel were able to work with a smaller, less resource-intensive version of the program that provides necessary functions. Also, central financial offices are able to maintain control over the definition and formatting of spreadsheet schemas, easing the task of merging data from numerous input sources in a consistent way. Thus a restriction of functionality actually resulted in a program configuration more useful than the old one in numerous contexts. However, in this case the overall functionality of the product was not reduced. Instead, the functionality was differently distributed within the various contexts of operation and use (the schema definers' context, the data entry context).

➤ Use of Analogy Domains

During *Scope Domain*, various analogy domains for the domain of focus may have been recorded in the DOMAIN INTERCONNECTION MODEL. During *Extend Domain Model*, these analogy domains can be revisited to identify analogous features supported in one domain but never carried over to the other. Analogy domains can thus be a source of ideas for both extending and restricting feature configurations.

➤ Extend stakeholders to new potential markets.

Look for market extensions based on the DOMAIN APPLICATION CONTEXTS domain relations described in the DOMAIN INTERCONNECTION MODEL. New aspects of existing markets may also be studied. These could include possible undocumented needs in the developers' or end-users' environments, suggested by particular feature innovations.

Look for related markets in terms of the CFRP-derived stakeholder role (i.e., consider pre- and after-markets, value-added resellers etc.). A risk here is expanding the focus of the domain too extremely.

> *Example.* In the Outlining domain, one application context considered in the DOMAIN INTERCONNECTION MODEL might be programmers using outline processors as simple forms of syntax-directed editors for viewing programs. If it takes some work to tailor an outline format useful for a particular language, it is possible that these outline templates or format files might become potential reusable assets in and of themselves. The domain boundary issue arises precisely at this point. Is the domain that of outline processing applications, or outline data that can be created using these applications?

➤ Map features to potential customer contexts

Use feature combinations for an exploratory search of potential customers for assets with the features described. Examine and expand the DOMAIN STAKEHOLDER MODEL to include potential ***customer contexts*** for reusable assets.

### Workproducts

■ EXTENDED DOMAIN MODEL

- EXTENDED FEATURE MODEL. Contains both precedented and unprecedented features and innovative extensions to features. The representation form used should be compatible with those used in earlier sub-phases. The content will be a superset of the INTERPRETIVE DOMAIN MODEL.

- EXTENDED STAKEHOLDER MODEL. Contains new semantic information about domain stakeholders, including attributes of potential contexts of application for features in the EXTENDED FEATURE MODEL.

## Guidelines

- Not all potential variants explored in this process need to be supported in any derived asset base. Bear in mind that adding combinations to the model is not a commitment to implement those features. This will enable the process to add quality and robustness assured by brainstorming, experimentation and rapid prototyping.

  Although innovation activities do not make binding commitments to the features to be supported by the asset base, some selectivity is still required in deciding which innovative features should remain in the final, extended domain model. For example, allowance must be made for applying some transformation steps that reach dead-ends in given situations.

## Exit Criteria

The EXTENDED DOMAIN MODEL can be considered complete enough for the transition to the *Engineer Asset Base* phase when:

- All relevant transformation rules have been applied consistently throughout the model and the results propagated back as needed throughout the various workproducts of the *Plan Domain Engineering* and *Model Domain* phases.
- The model has been *extended* to include features that simplify the semantics and coherence of the domain.
- Any features initially modeled within the scope of the domain that modelers have determined weaken the coherence of the domain model have been *excluded* through boundary adjustments to the various workproducts concerned.
- The resulting set of descriptive and innovative features have been configured into ***feature sets*** that represent well-designed subsets of domain functionality to be considered by asset base engineers. Each feature set has been correlated with a ***contextual profile*** that indicates attributes of application contexts where instantiations of domain functionality implementing the feature set would be viable.

## Validation and Verification

- Formally validate the EXTENDED DOMAIN MODEL with respect to the various closure properties corresponding to model transformation rules described in this section.
- Validate innovative features through additional interaction with domain practitioners.
- Reexamine practice in various domain contexts from the standpoint of novel feature variants. For features shifted to an alternative binding site, additional characterization of workflow in the new contexts might be needed to determine how useful such a feature would be. User requests for enhancements or modifications may have anticipated feature innovations.

# 7.0 Engineer Asset Base

## Purpose

The *Model Domain* phase of the ODM domain engineering life cycle has produced a DOMAIN MODEL that describes the range of variability for the domain of focus, a coherent space of possibilities for features in the domain. As described in Section 6.0 the DOMAIN MODEL can be used in many ways:

- as a direct source of codified domain knowledge;
- as a more formal basis for specifying or building individual systems in the domain; or
- as a basis for asset base addressing multiple contexts of application.

The primary purpose of the *Engineer Asset Base* phase of ODM is to scope, architect and implement an ASSET BASE that supports a subset of the total range of variability encompassed by the DOMAIN MODEL, a subset that addresses the domain-specific requirements of a specific set of customers.

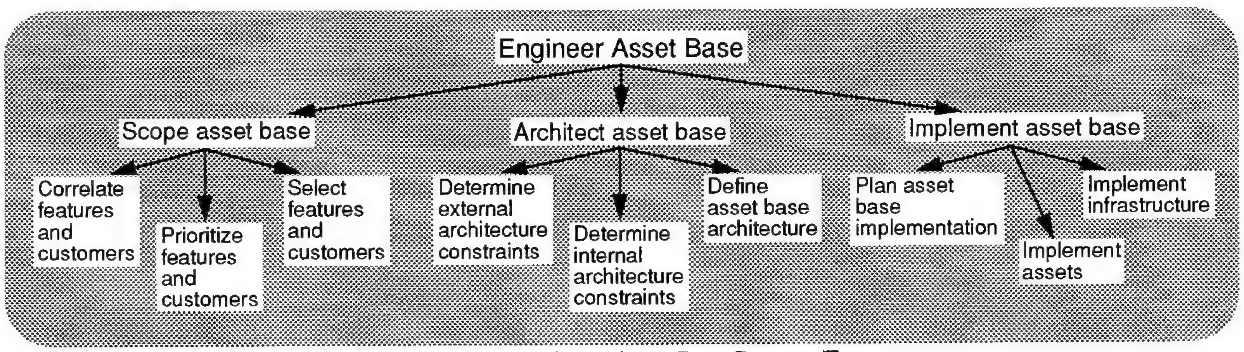

Engineer Asset Base
Scope asset base
Architect asset base
Implement asset base
Correlate features and customers
Prioritize features and customers
Select features and customers
Determine external architecture constraints
Determine internal architecture constraints
Define asset base architecture
Plan asset base implementation
Implement assets
Implement infrastructure

**Exhibit 53.** Engineer Asset Base Process Tree

The key benefit of the ASSET BASE produced in this phase of domain engineering is that it achieves an overall economy in the cost of developing systems in the domain, when viewed from the perspective of the customer contexts for the ASSET BASE taken as a whole. The key challenge of this phase is identifying an appropriate market of customers and the features required to support this market, to make the ASSET BASE a viable and ongoing structure for achieving increasing levels of reuse. An additional challenge in the *Engineer Asset Base* phase is to manage the potentially explosive variability in the domain in a process that eventually produces tractable specifications for implementors to create the asset base infrastructure and individual assets.

## Description

Where the *Model Domain* phase is primarily ***descriptive***, the *Engineer Asset Base* phase is ***prescriptive*** in nature. In this phase domain engineers commit to a specified range of functionality to be provided by assets developed/adapted for the asset base. Not all feature combinations noted during domain modeling need to be supported; some may not be of sufficient general use to warrant inclusion in reusable assets. Conversely, because of the *Extend Domain Model* task that concludes the *Model Domain* phase, features or feature combinations may be selected for the asset base that did not occur in any representative studied (i.e., in no existing system or known set of requirements for new systems). If the purely descriptive domain model describes the "as is", and the extended domain model described what "could be", the asset base model describes what "will be", that is, what is within that asset base's implementation scope.

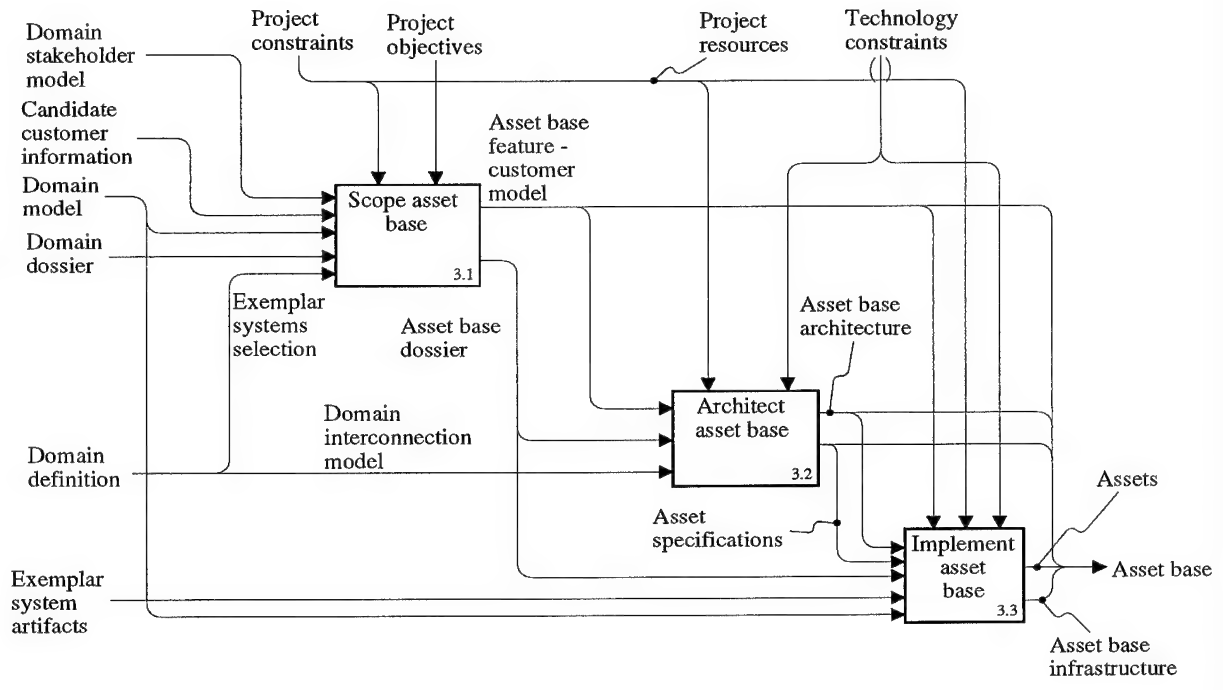


**Exhibit 54.** Engineer Asset Base $IDEF_0$ Diagram

The *Model Domain* phase produces a mapping from exemplar artifacts, to conceptual entities, to semantically characterized features. The *Extend Domain Model* task acts as the bridge between descriptive and prescriptive modeling by performing transformations on the INTERPRETED FEATURE MODEL itself.

As shown in Exhibit 54, the *Engineer Asset Base* phase consists of three main sub-phases:

- In *Scope Asset Base* domain engineers begin the task of directly specifying functionality to be supported by assets in the asset base. The focus accordingly shifts from ***domain informants*** to potential ***customers*** for the assets. Both the intended customers and the system architecture and feature variants to be supported in the asset base are selected in parallel. Feature and architectural variants are prioritized for usability relative to a specified set of potential asset customer contexts, and for feasibility relative to available asset implementation and management infrastructure.
- The primary goals of the second sub-phase, *Architect Asset Base*, are to produce an ASSET BASE ARCHITECTURE flexible and adaptable enough to cost-effectively cover the desired range of variability for the asset base; and to define this architecture to be evolvable and maintainable, minimally impacted by different implementation choices for different assets, and by evolution of assets over time.

  Selected features and feature configurations are mapped to high-level asset specifications. These specifications may reflect significant restructuring from the system artifacts first studied descriptively in *Model Domain.*
- The third sub-phase, *Implement Asset Base*, implements both the assets and the required

infrastructure for the asset base. A key challenge in this sub-phase is performing the trade-off analysis to determine the best implementation strategy (e.g., component- or generator-based techniques) for each asset and each configuration of assets that is likely to be accessed as a whole.

Not all domains require system architectural solutions. Vertical domains, encompassing large portions of an application, generally will require architected solutions. Horizontal domains, however, may tend towards more taxonomic and less component and interconnection structure in the asset base structure.

*Example.* Consider a domain engineering project focused on computer font data, including a taxonomy of typeface styles and operations upon fonts (e.g., condense, italicize, etc.) This is clearly a domain where useful results could be obtained without addressing conventional system architectural concerns.

Even for architecturally oriented domains, it should be possible to scope the *Engineer Asset Base* phase to exclude architectural issues; e.g., a domain engineering project could focus on producing a reference model of requirements or a suite of benchmark tests for applications in a given domain. Once again, while a structural notion analogous to software architecture (i.e., the "architecture" of a requirements specification or a test suite) is relevant here, the project may not need to focus on *architectural design* issues per se.

## Sequencing

- The ASSET BASE MODEL should remain a *subset* of the final DOMAIN MODEL. New feature variants or types of customer contexts should be migrated back to the INTERPRETED DOMAIN MODEL; and filtered through the *Extend Domain Model* task. The iteration might require backtracking as far as the DOMAIN DEFINITION.

  These should be considered exceptional conditions. There is a risk that iteration between the *Refine Domain Model* and *Scope Asset Base* sub-phases will result in an out-of-control process: i.e., an innovative feature is conceived, a potential customer is found, the customer context is studied, opportunities for other features are discovered, the innovation process continues, and so forth. In this way the set of features and proposed span of customer contexts could spiral out of scope.

  Without a certain degree of process traceability such breakdowns could be missed. One risk mitigation strategy is to strictly preserve the sequence of descriptive, interpretive, innovative, and prescriptive modeling stages, and to limit the feedback cycles allowed.

- The ASSET BASE ARCHITECTURE can reflect a *subset* of the ASSET BASE MODEL. Architectural constraints may rule out support for the full range of variability suggested in the feature and customer profiles in the ASSET BASE MODEL. Expect some iteration between the *Scope Asset Base* and *Architect Asset Base* sub-phases.

- Consider prototyping some individual ASSETS before completion of the ASSET BASE ARCHITECTURE. These should be considered prototypes, implemented as a risk reduction strategy in order to validate a candidate technology. While technology selection decisions are deferred where possible until the *Plan Asset Base Implementation* sub-phase, in practice the feasibility of applying some technologies will require some lead time to set up the infrastructure, train people and obtain required organizational commitment. Doing trial implementation as early as possible therefore makes good sense.

  Similarly, while the final ASSET BASE ARCHITECTURE should be validated against selected features in the ASSET BASE MODEL, trial architecture efforts can be initiated earlier to test the suitability of a particular architecture formalism or architecture description language.

- Architecture and technology selection may be parallel. The objective in the ODM process model is to encourage definition of a relatively technology-independent asset base architecture. In practice, architecture development and technology selection may be closely interleaved. Technology choices will require tuning and adjustment of the architecture; architectural choices may impose significant constraints on technologies selected.
- Multiple *Implement Asset Base* sub-phases. Once ASSET SPECIFICATIONS are determined, subsequent *Implement Asset Base* sub-phases can implement all or some of the specified ASSETS. In addition, asset utilizers can implement some assets for which there are only specifications.

## 7.1 Scope Asset Base

### Purpose

The primary purpose of this sub-phase is to derive an overall feature profile for the ASSET BASE and identify and characterize the ***market*** for the asset base, that set of application contexts in which practitioners will potentially utilize domain assets. The key function of this sub-phase is to derive a subset of the features and potential customer contexts described in the DOMAIN MODEL mapped to a specific set of customers for the asset base.

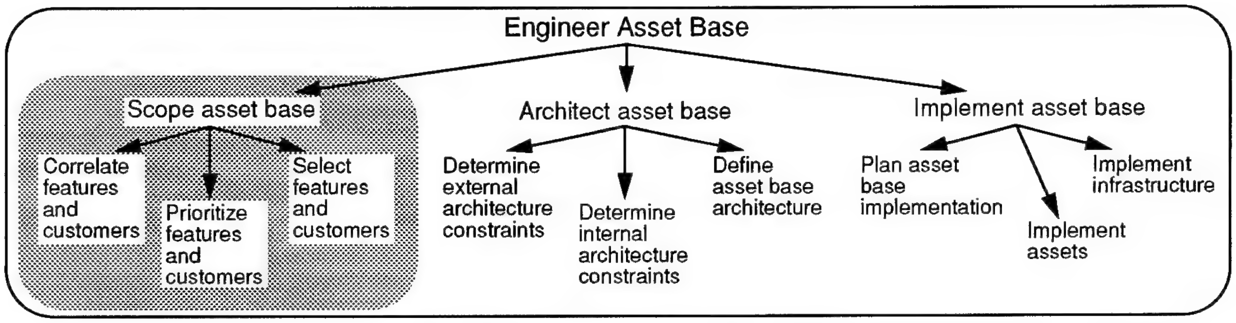

Engineer Asset Base
Scope asset base
Correlate features and customers
Prioritize features and customers
Select features and customers
Architect asset base
Determine external architecture constraints
Determine internal architecture constraints
Define asset base architecture
Implement asset base
Plan asset base implementation
Implement assets
Implement infrastructure

**Exhibit 55.** Scope Asset Base Process Tree

A key challenge of this sub-phase is the fact that both the features and the customers to be supported are dynamic, and the decisions are inter-dependent. Another challenge is the fact that the correlation of features to customer contexts is initially based on the descriptive correlations in the DOMAIN MODEL; hence there is considerable uncertainty in how well-matched the correlations are to the real contexts in which reusable assets must be adopted.

The *Scope Asset Base* sub-phase can be compared to the requirements analysis phase of conventional software engineering. The resulting ASSET BASE MODEL specifies what features must be supported by the asset base; in subsequent sub-phases, the ASSET BASE ARCHITECTURE and ASSET BASE IMPLEMENTATION PLAN will embed specific technology choices that determine how these feature requirements will be satisfied.

However, this sub-phase also differs significantly from the requirements analysis phase in conventional software engineering in that:

— features, unlike requirements, can impose constraints on design and even implementation details; they are not just high-level requirements;

— features within a feature set can conflict, and features across feature sets can be redundant (i.e., feature sets are not ***partitionings*** of the total range of features in the same way that a requirements allocation partitions a set of system requirements); and

— there is no pre-ordained single customer to whom all features apply.

### Description

Domain modeling has produced set of domain features, a mapping of stakeholder interests to those features, feature interdependencies and, possibly, feature profiles of certain domain artifacts that may become candidates for reengineering into assets. The task now is essentially one of simultaneous constraint satisfaction. A set of features to be supported by the asset base must be specified. A set of stakeholders whose use of the asset base is anticipated and supported, the potential ***asset base customers*** (hereafter referred to simply as "customers"), must also be selected. The driving force on convergence is matching potential ***customer contexts*** with feature combinations that are candidates for inclusion in the ASSET BASE MODEL.

As shown in Exhibit 56, *Scope Asset Base* consists of three main tasks.

- The *Correlate Features and Customers* task maps the feature sets and their associated profiles of potential contexts of application to specific customers selected from the DOMAIN STAKEHOLDER MODEL.
- The *Prioritize Features and Customers* task evaluates the anticipated benefit of each feature set within the market, and the anticipated level of effort to implement and maintain the feature set.
- The *Select Features and Customers* task marks the clearest transition from descriptive to prescriptive modeling, from the realm of possibilities to that of commitments.

At the end of this sub-phase, there should be traceability links demonstrating consistency across these two sets of scoping decisions, i.e., between anticipated asset customers and supported features, and between architectural and more fine-grained features to be supported.

## Sequencing

This sub-phase can be performed in either a ***feature-driven*** or ***customer-driven*** sequence, or some combination thereof. The feature-driven approach works "outward" from identified sets of features, to associated potential contexts, to specific customers that match the profile of those contexts. Alternatively, if there are definite ***core customers*** mandated or strategically determined for

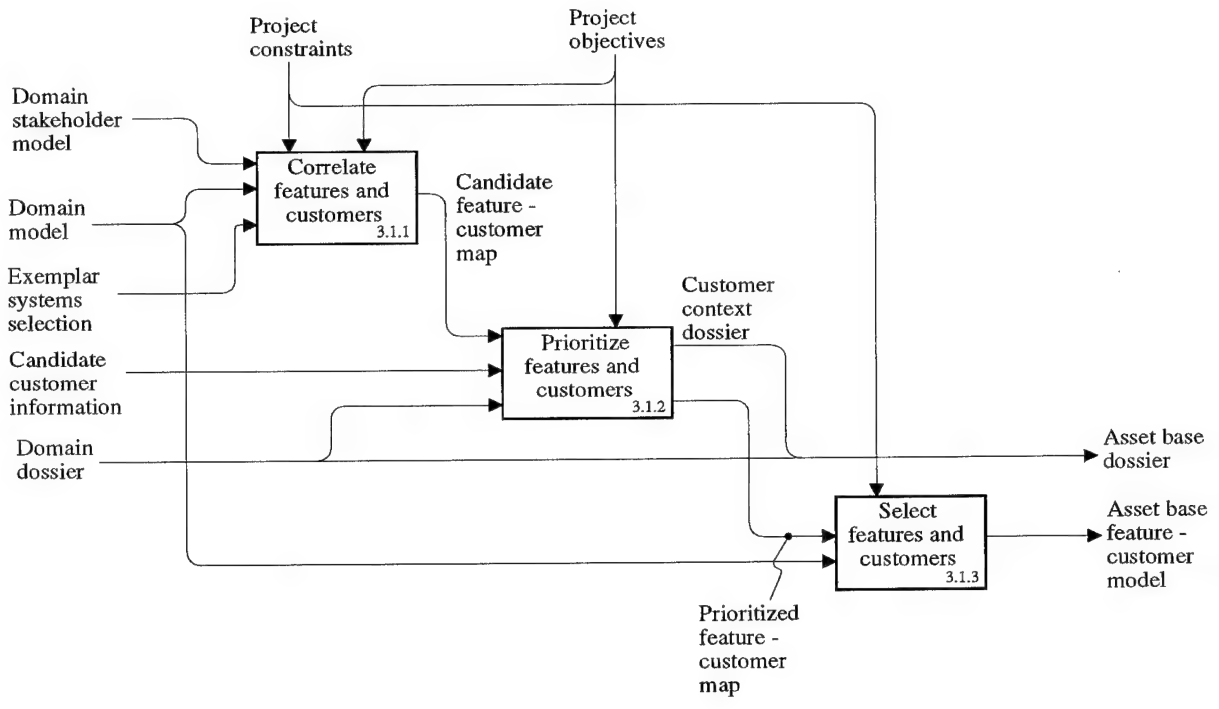

Project constraints
Project objectives
Domain stakeholder model
Domain model
Exemplar systems selection
Candidate customer information
Domain dossier
Correlate features and customers 3.1.1
Candidate feature - customer map
Prioritize features and customers 3.1.2
Customer context dossier
Asset base dossier
Select features and customers 3.1.3
Asset base feature - customer model
Prioritized feature - customer map

**Exhibit 56.** Scope Asset Base IDEF$_0$ Diagram

the project, these customers can be used to establish a corresponding set of ***core features*** to support.

Selection of prescriptive features can also be performed in two general ways, using a ***bottom-up feature-driven*** vs. a top-down, ***architecture-driven*** approach. The bottom-up feature-driven approach describes feature variants to be supported for individual asset-level functionality. The architecture-driven approach explores feature sets describing the domain system as a whole.

Trade-off analysis may iterate between bottom-up analysis of individual features and feature variants, and top-down analysis of feature profiles for entire domain systems (i.e., systems or subsystems representing the full scope of the domain of focus, rather than individual features within the domain).

## 7.1.1 Correlate Features and Customers

### Purpose

The final output of the *Model Domain* phase was a DOMAIN MODEL that included a number of overall ***feature sets***, each representing a coherent segment of domain functionality, and characterized in relation to other feature sets and to ***potential market profiles***. The first task within the *Scope Asset Base* sub-phase of *Engineer Asset Base* is to correlate these feature sets and market profiles to specific customers. The primary result will be a CANDIDATE FEATURE — CUSTOMER MAP in which feature sets that are viable candidates are identified and correlated to ***candidate customer contexts*** within the overall stakeholder context for the domain and the project.

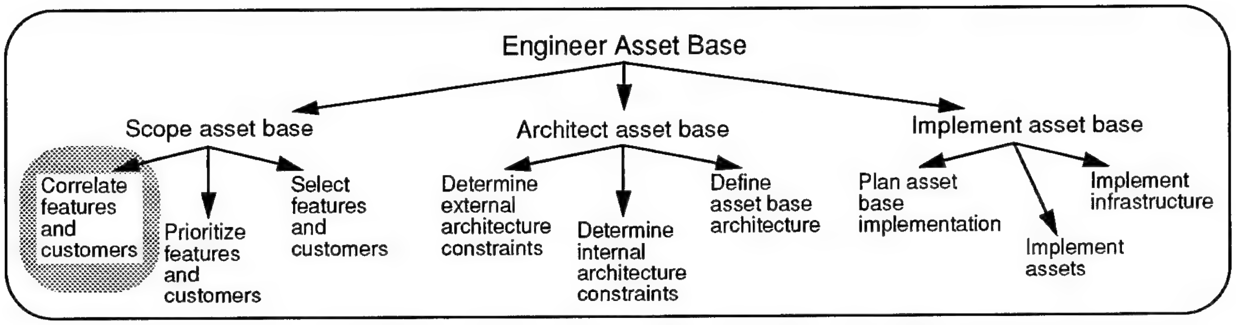

Engineer Asset Base
Scope asset base
Architect asset base
Implement asset base
Correlate features and customers
Prioritize features and customers
Select features and customers
Determine external architecture constraints
Determine internal architecture constraints
Define asset base architecture
Plan asset base implementation
Implement assets
Implement infrastructure

**Exhibit 57.** Correlate Features and Customers Process Tree

The primary purpose of this task is to provide an opportunity to reestablish an overall organization context for the *Engineer Asset Base* phase of the project. Assumptions about the context for the ASSET BASE that were made at the start of the project, in *Plan Domain Engineering* activities, are almost certain to have changed over the course of the *Model Domain* phase. In fact, one criterion of success for this phase is that these assumptions will have changed. Domain practitioners have participated in the modeling effort and should perceive possibilities for reusable assets in different terms; modelers should more thoroughly understand the limitations and possibilities of the domain.

As a result, the relevant context for the DOMAIN MODEL may be quite different than for the ASSET BASE. In fact, multiple asset bases could be engineered using a common DOMAIN MODEL as a basis. In order to reap the benefits of this new situation, it is a critical element of the ODM process to perform explicit re-contextualizing activities as part of this task.

In this respect, the *Correlate Features and Customers* task is analogous to the early context-setting activities in the *Plan Domain Engineering* phase, i.e., the *Determine Candidate Project Stakeholders* task within the *Set Project Objectives* sub-phase. In both these tasks, project planners and asset base modelers respectively perform a descriptive scan of the possible stakeholders, deferring to later tasks the final selection and commitment. However, the actual process performed is quite different in this task, because it builds on an already completed DOMAIN MODEL. Trade-off analysis and selection of which feature sets to support and which markets to target as explicit customers are performed in the subsequent tasks of this sub-phase, *Prioritize* and *Select Features and Customers*.

## Entrance Criteria

- There is a complete and validated DOMAIN MODEL for the domain of focus. In particular, the number of possible feature variants must have been reduced through constraints documented in the INTERPRETIVE DOMAIN MODEL and transformations of features incorporated into the EXTENDED DOMAIN MODEL.
- There is an ***initiative*** to proceed with the *Engineer Asset Base* phase of domain engineering.

  This criterion allows for the possibility that some delay may ensue between completion of the DOMAIN MODEL and the *Engineer Asset Base* phase. Much of the care that goes into the completion and validation of the DOMAIN MODEL is to assure its viability over time. Also, as mentioned in the Purpose sub-section above, multiple *Engineer Asset Base* efforts could be initiated starting from a common DOMAIN MODEL.

## Inputs

- DOMAIN STAKEHOLDER MODEL. The source for candidate customers for the asset base.

  The task works with *domain* and not the original *project* stakeholders. Project stakeholders without explicit interest in the domain results are still involved and reflected in the controls on this task, i.e., PROJECT OBJECTIVES/CONSTRAINTS.

  The task works with all domain *stakeholders* and not only domain *informants*, that subset of stakeholders who were the focus of interactions in the *Model Domain* phase. The aim is now to cast the net as wide as possible: some informants will become customers; but many potential customers will not have been consulted as informants. How well the feature sets within the DOMAIN MODEL meet the needs of these other customers will be one significant validation of the quality of the sample selected in the *Plan Data Acquisition* task of the *Acquire Domain Information* sub-phase.
- DOMAIN MODEL. The source for candidate feature sets for the asset base, as well as for the associated market profiles that guide identification of specific candidate customers.
- EXEMPLAR SYSTEMS SELECTION. Systems that intersect the domain, from which candidate customer systems will be drawn in this task.

## Controls

- PROJECT OBJECTIVES. Overall project objectives, specific objectives for the *Engineer Asset Base* phase that were identified earlier in the project, and any modified or new objectives, possibly a result of or reaction to earlier project phases.
- PROJECT CONSTRAINTS. There may be new constraints, including constraints created by pre-

vious phases of the domain engineering life cycle; e.g., success in the *Model Domain* phase may engender expectations for the *Engineer Asset Base* phase of the project.

## Activities

➤ Identify initial list of candidate asset base customers

Working from the DOMAIN STAKEHOLDER MODEL, identify those stakeholders with potential interests specifically in the ASSET BASE to be developed. Using the PROJECT OBJECTIVES, identify those who are ***key customers***: i.e., project success depends on satisfying their requirements as asset base customers. Then identify any other stakeholders with a potential interest in the asset base. For all candidates, indicate the ***stakeholder interests*** that seem relevant to requirements for the asset base. Document the results in an interim list, distinguishing key and optional customers and their interests (not illustrated by a template).

These may be dynamic and time-sensitive relations; this highlights the importance of this task being performed close to the time when asset base development will begin.

➤ Identify initial list of feature sets

Working from the EXTENDED DOMAIN MODEL component of the DOMAIN MODEL, identify specific feature variants and/or feature sets that seem viable for consideration. This could include the entire model if there are no semantically significant ways of excluding feature sets. For large and complex models, the PROJECT CONSTRAINTS should provide some filtering criteria; e.g., global contextual constraints that exclude certain feature variants and combinations. The resulting subset of the DOMAIN MODEL will be further transformed in this task. Summarize the major feature sets in an interim list (not illustrated by a template).

➤ Profile candidate customers

Using the potential market profile associated with each candidate feature set derived from the EXTENDED DOMAIN MODEL, *scan* the candidate asset base customers for degree of matches to the profile. In effect, this is classifying the customers according to market features or attributes. Record the results in the CANDIDATE FEATURE — CUSTOMER MAP, as described in the Workproducts section below. A template of this workproduct is illustrated in Exhibit C-13.

The mapping can be performed working from a core feature set (i.e., defining features for the domain) to potential customers, then back to feature sets; or vice versa. The mapping can also be done in one pass or iteratively.

## Workproducts

■ CANDIDATE FEATURE — CUSTOMER MAP

This workproduct shows the anticipated degree of interest that candidate customers have for candidate feature sets derived from the DOMAIN MODEL. Each feature set is a subset of the features in the DOMAIN MODEL. Each customer market segment is a segment of the DOMAIN STAKEHOLDER MODEL Both elements are still *candidates* only. Final selections are made in subsequent tasks.

The mapping is *many-to-many*: Each feature set may be deemed of interest to multiple customers. Conversely, each customer may be interested in multiple feature sets.

- For feature sets that are ***components*** of the domain, the more components that interest a given customer the more likely it is that the customer will utilize multiple assets in subsystem level aggregations.
- For feature sets that are ***specializations/generalizations*** of the domain, a customer's interest in more than one feature set indicates the degree of ***variability*** that is required in the ASSET BASE.

A template for this workproduct with more detailed procedures is included in Exhibit C-13. To the degree that the feature sets can be ordered in strict subset relation, redundancy in listing the attributes for each feature set can be avoided in this workproduct.

## Guidelines

- Treat stakeholders as candidate customers. In the *Model Domain* phase it was important to treat stakeholders primarily as informants, even though many informants are potential customers as well. In the *Engineer Asset Base* phase, domain engineers need to treat stakeholders primarily as customers, even though their help as informants may still be required at various points in the process. Having said this, however...
- Don't try to meet the full spectrum of any specific customer's needs. The asset base need not provide all system requirements for any potential customer context; nor even all *domain* functionality required in these contexts. The asset base must provide sufficient domain coverage to motivate application engineers to utilize assets in developing systems. Aspects of domain functionality may be left to application engineers to implement, if a clean interface from asset-supported to developer-customized components can be provided.

## Exit Criteria

- All key customers have been identified based on PROJECT OBJECTIVES and PROJECT CONSTRAINTS.
- The set of features and customers that will be analyzed in more detail has been filtered as much as possible to minimize the subsequent selection effort.
- Each feature set identified in the EXTENDED DOMAIN MODEL has been mapped against each candidate customer.

## Validation and Verification

- Formally validate the CANDIDATE FEATURE — CUSTOMER MAP for accuracy, completeness and consistency. Update previous workproducts as necessary; for example, if new attributes are discovered that are useful to the market profiles in the DOMAIN MODEL, these should be handled by iterating back to the *Extend Domain Model* task.
- Empirical validation of the correlations in the CANDIDATE FEATURE — CUSTOMER MAP, which are still somewhat hypothetical, is performed as part of the next task, *Prioritize Features and Customers.*

## 7.1.2 Prioritize Features and Customers

### Purpose

In the *Extend Domain Model* task, modelers searched for potential contexts of use for given features. In the *Correlate Features and Customers* task, these potential market profiles for features were *grounded* in a set of specific candidate customers. The resulting CANDIDATE FEATURE — CUSTOMER MAP provides an initial correlation of features to these customer contexts. In order to make a strategic selection of the features to support and the customers to target for the asset base, the *Prioritize Features and Customers* task now attempts to find answers to the following questions:

— How valuable would a given feature set be to a given customer? Would reuse be at a coarse or fine-grained level? Individual components or entire sub-systems?

— How many instances of use would be likely for each customer context? How cost-effective would reuse be for the customer? How much would they pay?

— How feasible would it be to implement and maintain a given feature set?

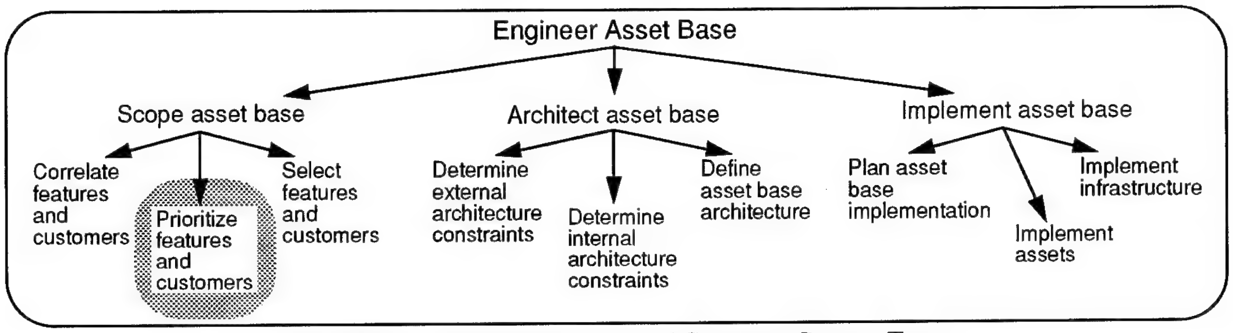

Engineer Asset Base
Scope asset base
Architect asset base
Implement asset base
Correlate features and customers
Prioritize features and customers
Select features and customers
Determine external architecture constraints
Determine internal architecture constraints
Define asset base architecture
Plan asset base implementation
Implement assets
Implement infrastructure

**Exhibit 58.** Prioritize Features and Customers Process Tree

The principal benefit of separating this task from its predecessor, *Correlate Features and Customers*, is that it helps to distinguish activities focused around re-establishing the stakeholder context from activities focused around asset base design decisions. This methodological step helps mitigate the dual risks of determining asset base requirements on the basis of one dominant customer; or on the basis of no explicit customer at all.

The principal benefit of separating this task from its successor, *Select Features and Customers*, is that it helps to distinguish activities focused around evaluation and weighting from final selection. This allows for a certain amount of "what-if" analysis, forecasting, and validation with candidate customers before closing the decision process.

A key challenge in this task is establishing a valid empirical basis for usability and feasibility assessments. Features, by their nature, are transformed significantly from concrete implementations; this means that there is no simple way to survey the potential number of instances of application. Similarly, it is difficult to assess feasibility without committing to or assuming a particular implementation technology or architectural approach.

### Entrance Criteria

- The initial version of the CANDIDATE FEATURE — CUSTOMER MAP has been completed. This serves as a filter to reduce the complexity of data to be analyzed.

## Inputs

- CANDIDATE FEATURE — CUSTOMER MAP. Developed in the preceding task, *Correlate Features and Customers*, this workproduct is refined in this task to incorporate more detailed information about usability and feasibility of various feature sets.

- DOMAIN DOSSIER. A source for information about legacy artifacts that may affect feasibility and/or level of effort to implement certain feature profiles; and constraints of customer contexts that were analyzed as part of the *Model Domain* phase.

- CANDIDATE CUSTOMER INFORMATION. Supplemental information from potential customers about feature preferences and priorities, used for the trade-off analysis performed in this task. CANDIDATE CUSTOMER INFORMATION includes configurations of features that must be available as an ensemble and features that must be excluded from ensembles for the ensemble to be usable to a specific customer.

## Controls

- PROJECT OBJECTIVES. Needed to guide trade-off analysis; particularly to establish what degree of thoroughness is required, based on project expectations.

  *Example.* If the PROJECT OBJECTIVES specify that the project should demonstrate a factor of two reduction in development costs for application groups that become asset base customers, this might lead to prioritizing customers with potential for large-grained reuse over lower levels of reuse by more customers.

## Activities

In the *Prioritize Features and Customers* task, asset base modelers refine the CANDIDATE FEATURE — CUSTOMER MAP to facilitate a cost-benefit style analysis of different feature sets and different asset base markets. More resources may be required to implement an expanded feature set, or to engineer a given feature set to meet the variability requirements for an expanded set of customers. The final selection is made in the succeeding task, *Select Features and Customers*.

➤ Select supporting methods

***Market analysis*** methods are an essential component of the *Prioritize Features and Customers* task. In the ODM context, this is the point in the life cycle where market analysis techniques are most directly applicable. In this task, asset base modelers assess the *demand* for various specific feature variants and overall system configurations within given *market segments*. See Section 8.8 for guidelines on integrating market analysis methods with ODM processes.

The parallels are striking enough that some researchers have described domain analysis as a whole as market analysis specialized to software components. However, there are some important caveats to be considered in applying these techniques.

a) Conventional market analysis is organized around *new product development.* An important aspect of the ODM approach to domain engineering is capitalizing on legacy technology. This is a different process in many ways from product development per se.

b) Conventional market analysis is oriented toward *manufactured goods*. Domain engineering, at least in the scope of this Guidebook, is oriented around software or information reuse. Many assumptions about packaging, distribution, economies of manufacture, etc., that are implicit in market analysis techniques simply don't work the same way in the software industry. The high degree of variability obtainable in software is a related complicating factor.

c) The target market of an asset base, and the immediate customers for domain engineering, are *applications developers*. This means that at a minimum the domain engineer must consider two separate ***customer contexts***, the immediate (i.e., developer) and indirect (i.e., end-user) customer contexts. This introduces still other elements to the nature of the task.

The remaining Activity sub-sections will describe essential elements specific to performing this analysis in the domain engineering context.

➤ Characterize candidate customer contexts

Characterize candidate customer contexts to provide supplementary information to guide asset base scoping and asset implementation decisions. This may involve gathering additional information about contexts already documented in the DOMAIN DOSSIER or studying new contexts. These contexts might now be studied to determine whether the features deemed of interest are in fact of interest to those practitioners.

While superficially this may appear like a return to data acquisition, the process followed is quite different than that employed during the *Model Domain* phase. Here, the model of features has been created and the information-gathering is more in the nature of a focus group or market analysis. The focus is on validating the linkage between features and anticipated customer interests. The basic question being asked is: If you had an asset satisfying this set of features available, would you use this asset rather than implementing the capability from scratch?

In some cases it may be determined that a given set of features requires augmentation with certain other features before an asset supporting those features would get used. Often there will be large sets of functions that must be provided as a whole; e.g., few people would use a single X-window routine out of a library. In other cases it is possible that a feature might need to be *excluded* from a feature set in order to gain acceptance.

*Example*. In the Outlining domain, suppose that the EXTENDED DOMAIN MODEL defines a variant configuration that strips away editing functions, preserving only outline read-only browsing capabilities. Users of help systems for on-line applications have been identified as a candidate customer context for this restricted variant of outlining capabilities, a class of application contexts not studied during the *Acquire Domain Information* sub-phase.

➤ Acquire supporting data

More information may be needed for a number of customer contexts, even for representative systems studied in the *Model Domain* phase. Document any new information acquired in the CUSTOMER CONTEXT DOSSIER.

*Example*. Assume that a number of requirements specifications, following a particular standard, were examined during the *Model Domain* phase. The DESCRIPTIVE MODELS may have noted certain features of these requirements documents, including their internal structure. Now, in prescriptive modeling, we must consider whether the asset base will itself contain requirements text and/or requirements documents (or document templates, fragments, generators, etc.). If so, we will probably require a finer level of detail about the use of such documents within the various customer contexts.

➤ Quantify usage estimates

Prioritize feature profiles according to the estimated frequency and distribution of uses of the asset within the marketplace. Document the core or typical context of most probable use, as well as more peripheral opportunities.

There needs to be *sufficient* potential use for a feature variant to warrant its inclusion in the feature profile for an asset. One approach to estimating the "market share" of feature variants within potential contexts of use might be the following:

- Across instances of use, come up with estimated scale of cost-to-adapt per instance of use;
- Characterize each variant across its instances of use using cost-to-adapt estimate;
- Allow for instances excluded or enabled by a given variant as well as straight comparison.

  *Caveat*. Estimating the potential market for a set of features within a set of potential customer contexts is not an exact science. Current methods and automated support for generating such quantitative estimates are limited. This is in part because reusable assets may employ generative as well as component-based techniques; thus potential contexts of application might not be readily apparent from code-based analysis.

➤ Analyze feasibility

Analyze feature combinations and architectural variants from the standpoint of feasibility. Consider the level of effort required both to implement the features and to maintain and do configuration management on separate variants within the asset base. Some feasibility issues to consider include the following:

- Functionality/performance trade-offs. Support of variability always carries a price, whether in performance, asset base overhead, complexity of the asset search and retrieval process, or increased complexity in configuration management and asset maintenance.
- Uncertainty in estimation. Cost of implementation for features implemented in previous systems should be more predictable than for novel features. If project metrics are available for exemplar system development efforts, levels of effort for implementation and maintenance of artifacts implementing desired features may provide some guidelines. Clearly, however, such estimates will be far from an exact science. Estimates will be even more difficult for novel feature variants. Additional input from domain experts and experienced asset developers will probably be necessary.
- Technology assumptions. Although some working assumptions must be made here about the implementation technology to be used, the purpose here is not to make the technology choice, merely to gain additional information for the trade-off analysis of selecting features to be covered in the asset base. As much as possible, avoid preemptive commitment to particular implementation techniques. Feasibility assessments will need to be revisited after asset implementation choices have been made. Closely related variants may appear to necessitate independent and separately maintained code components; however, if generative techniques are to be used the variants may be derivable (and maintainable) from a single source work-product.
- Configuration Management. Consider configuration management (CM) issues as part of maintainability issues. Providing separate variants can be compared to the CM problem of a system where multiple versions are simultaneously fielded and must be managed in parallel.

➤ Analyze usability and feasibility trade-offs

Synthesize the results of usability and feasibility analysis; this involves consideration of packaging trade-offs analogous to those dealt with in product line planning. Validate that distinct feature clusters have sufficient "breathing room" across the intended range of customers to warrant the overhead of separate creation and configuration management. This reflects a general rule in designing for reuse: the reusability of each asset must be considered within the context of the asset base within which it will reside, not only in isolation. (See Exhibit 59.)

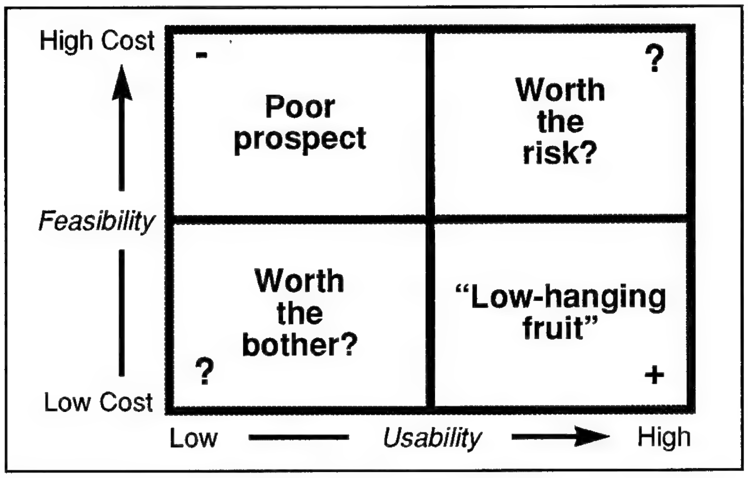


**Exhibit 59.** Usability-feasibility trade-offs

Some key trade-off issues are discussed in the following paragraphs.

- Product line segmentation. One trade-off to be considered is analogous to that in product family packaging decisions: ensuring that variants supporting particular combinations of features have sufficient demand across the intended range of customers. Assess the anticipated flexibility of different customers with respect to feature profiles optimal for their needs. Also assess the tolerance of potential customers for small, incremental variation in assets. Supporting too many variants that differ in subtle ways irrelevant to users may actually weaken the usability of the asset base. Thus there is a inherent trade-off from the standpoint of usability. This discusses usability aspects of this trade-off; feasibility aspects are discussed below (e.g., the overhead of separate creation and configuration of closely related variants).

- Feature "distance" between variants. The value of a given asset within the asset base is determined in part by its ***feature distance*** from related assets. Formally, a distance measure can be defined for any two feature variants in terms of their relative position within a feature model. This measure can be extended to feature profiles, as some function of the distance measures of the constituent features. Since a feature profile can be treated as a specification for a potential asset, the concept of feature distance provides a somewhat formal way of characterizing distances between potential assets, i.e., in terms of their feature profiles.

  *Example.* In the Outlining domain, consider an outlining program that allows up to 9 heading levels, versus one that allows 15 heading levels. The question is: for a developer who wanted 15 heading levels, how much value does that variant provide relative to the 9-level variant? This value might be worth supporting in the asset base; it might require a simple change of parameter, or may have more pervasive impact on issues like screen display, memory allocation, etc.

  By contrast, would it be worth separately providing a 9 versus a 10 level heading version? This difference probably would not either significantly change the level of interest for a particular customer, or greatly expand the market of customers for the asset base.

- Customer flexibility. An asset need not satisfy an application engineer's entire set of requirements, even within the domain scope. It must satisfy enough of the requirements to motivate its use, and must result in a perceptible cost-effectiveness and/or productivity improvement for the engineer. Robust mechanisms for integrating the asset into applications can encourage some flexibility in requirements.

  *Example.* In the example above, could a given application tolerate fewer heading levels? For writers or idea processor users, this might be acceptable. For an application involving using an outline interface to display directory structures, on the other hand, this limit might be unacceptable. In this example, would it be simple or complex for the engineer to add extensions to the asset to obtain the extra functionality needed? Conversely, what if there were a desire to restrict the number of available heading levels below the system maximum provided?

## Workproducts

■ PRIORITIZED FEATURE — CUSTOMER MAP

Usability and feasibility estimate for each feature set and correlated market segment. This workproduct is a refinement of the FEATURE — CUSTOMER MAP produced in the previous task, *Correlate Features and Customers.*

■ CUSTOMER CONTEXT DOSSIER

For those interfaces identified in the DOMAIN INTERCONNECTION MODEL, and those customer systems that were originally part of the REPRESENTATIVE SYSTEMS SELECTION studied during the *Model Domain* phase, relevant features are carried over from the DOMAIN DOSSIER into the CUSTOMER CONTEXT DOSSIER. Any other information about customer context is captured in this dossier as well. The CUSTOMER CONTEXT DOSSIER is used in determining the external and internal constraints on the ASSET BASE ARCHITECTURE in the *Architect Asset Base* sub-phase.

## Guidelines

- Usability profiles need not be linear. More is not always better. In mapping feature sets to given customer contexts, increased functionality will typically reach a peak, then decrease rapidly in terms of perceived usability.

## Exit Criteria

- Sufficient supplementary data has been acquired about customer contexts and feasibility issues.
- Estimates are documented of the usability and feasibility factors for each feature set with respect to each customer context.

## Validation and Verification

- Check estimates for internal consistency (allowing for the non-linearities discussed above in Guidelines). If similar customer contexts produce wildly different estimates, look for the explanatory factors and allow for error in the estimation process.

- Obtain feedback and review from prospective customers. Provide them with specifications of individual assets or overall system configurations derivable from proposed variant architectures. Their reactions will serve as early validation as to whether the feature combinations and system variants supported are deemed useful by application engineers.

  Note constraints or caveats potential utilizers mention about the required form, structure, or other characteristics for assets. Where possible, though, suppress details about how assets will be implemented that might influence customers' assumptions about the usefulness of a given variant.

## 7.1.3 Select Features and Customers

### Purpose

The first two tasks in the *Scope Domain* sub-phase produce a great deal of information about the potential value of various feature sets to various candidate customers. The primary purpose of this task is to make the final selection of features and customers for the ASSET BASE; these decisions are recorded in the ASSET BASE MODEL.

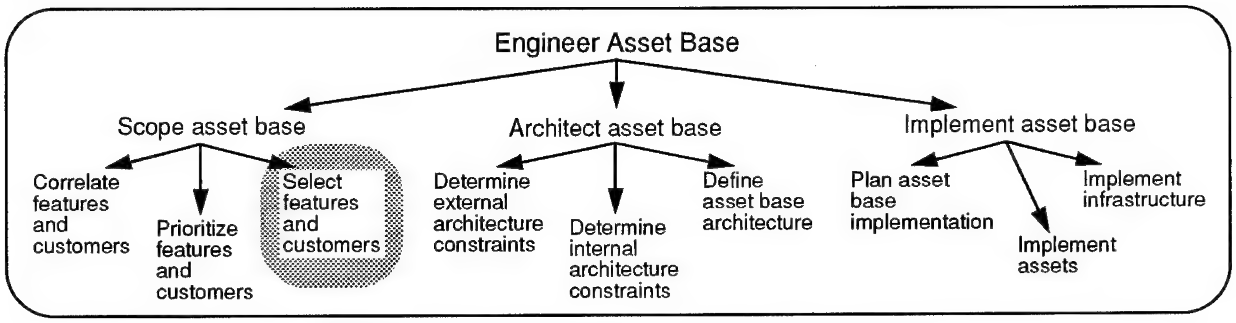

Engineer Asset Base
Scope asset base
Architect asset base
Implement asset base
Correlate features and customers
Prioritize features and customers
Select features and customers
Determine external architecture constraints
Determine internal architecture constraints
Define asset base architecture
Plan asset base implementation
Implement assets
Implement infrastructure

**Exhibit 60.** Select Features and Customers Process Tree

This task represents the fulfillment of one of the primary objectives of ODM, which is to facilitate the creation of specifications of reusable assets that make clear not only their functionality (via the feature profile) but their *intended scope of applicability* (via the customer-related portions of the ASSET MODEL. By selecting the features and customers and documenting the linkage, the rationale for each is integrated into the model. This rationale provides valuable information downstream for the implementor of the ASSET BASE INFRASTRUCTURE, who will need to support potential utilizers searching for assets; they may be best able to characterize what they need in terms of their own customer profile, which can be mapped to the feature profile using, in part, information generated in this task.

There are several important aspects to this task within the context of the overall process model.

- Features and customers are selected in parallel, rather than in a sequence driven by one factor over the other. Maintaining this balance between the two, while a difficult decision procedure, helps ensure the overall quality of the asset base; it makes it more likely that there will be a coherent market of customers specified for the asset base, as well as a coherent range of features supported.

- Decisions on features to be supported precede architectural decisions about how to support these features in the asset base.

Note: this is a point of possible confusion, because the features themselves may contain requirements for supporting particular *system architectures* in the assets and the applications to be built using assets. However, what is *not* decided within this task is how the asset base will deliver components in the proper configuration.

*Example.* In the Outlining domain, the ASSET BASE MODEL may specify that two versions of the outliner asset must be supported, one that is cleanly separated from the host word processor program, communicating with an event-driven interface; the other, a system that can be tightly integrated with specific word processors based on some customization/reconfiguration process. Each of these represents decisions roughly correlating to *system architecture* decisions. The *asset base architecture* decisions still deferred would include issues such as: how much sharing between code components will take place between the two versions of the program?

## Entrance Criteria

You are ready to *Select Features and Customers* for the ASSET BASE when:

- The customers' organization contexts for the ASSET BASE have been identified and mapped to the DOMAIN MODEL in *Correlate Features and Customers.*

  Deciding on features and customers for the asset base without first performing this task means that any assumed connections between customers identified and features to be supported are implicit only. There is no way to validate and verify the connections, and therefore no way of checking whether viable choices have been made.

- The correlations have been refined by assessing the potential usability and feasibility of each feature set, relative to identified customers.

  Deciding on features and customers for the asset base without first performing this task means that the decisions may be skewed by the minimal correlation first produced. So, for example, a feature set of interest to several customers might be preferred to one of interest to just one customer, when in fact the potential instances of use for the former are much fewer than for the latter.

## Inputs

- PRIORITIZED FEATURE — CUSTOMER MAP. The candidate features and customers to be selected in this task.
- DOMAIN MODEL. Supplemental information to the primary input above.

## Controls

- PROJECT CONSTRAINTS. Estimated level of effort for implementing various feature sets are mapped to constraints such as budget, schedule, available staff and resources, etc. By deferring consideration of these constraints to this task, the PRIORITIZED FEATURE — CUSTOMER MAP produced in the previous task can be reused if project resources expand.

## Activities

➤ <u>Make selection and document</u>

Based on project resources

Document final decisions about which features the asset base will support in the ASSET BASE MODEL. This model represents the binding commitments or specification for the overall feature capabilities of assets within the asset base. It should be a subset of the features in the DOMAIN MODEL. It can be represented in various ways, such as a restriction or pruning of the DOMAIN MODEL. The ASSET BASE MODEL is termed a model because it should retain all semantic information relevant to selected features. However, the model is created through strategic transformation and filtering of the previous model, *not* by an analogous process.

Consider the selection process in terms of one feature. One option is simply to select a single variant as the variant to be supported in the asset base. Other variants are excluded from the ASSET BASE MODEL; asset engineers need not address how to obtain them. Another option is to implement both variants, even if both could not co-exist in a single run-time system. The asset implementor might develop both variants and store them in the asset base, or implement a generator that can create both variants as instances, to be generated on request by asset utilizers or generated and archived. Some features or contexts of use within the original descriptive scope of the domain model may be excluded from the ASSET BASE MODEL. Typically, the reason is insufficient need for that functionality in the anticipated contexts of use.

Where architectural variants have been rejected as part of the ongoing asset base, consider these variants as incremental builds in a phased development plan for the asset base. This information will feed forward into the detailed planning activities of the *Plan Asset Base Implementation* task.

➤ <u>Determine assets of interest</u>

Select the set of assets likely to be desirable end results of the *Engineer Asset Base* phase. An asset can be considered any component, tool, or other resource that domain practitioners do their work more effectively. Consider the following forms that assets can take:

- <u>Artifact-based assets</u>. Artifact-based assets include traditional ***components*** that will be retrieved by application developers, possibly adapted or modified, and then incorporated into their own products. Some comparative sampling of artifacts of a given type should have been done as part of the *Acquire Domain Information* sub-phase of the *Model Domain* phase. This information is available in the DOMAIN DOSSIER. Other candidates for reengineering may be identified by asset developers.

  Other artifacts that are candidate assets are interim artifacts used in system development but not part of the deliverable system. Examples include requirements checklists used by analysts as reminders, test data, debugging scripts, etc.

- <u>Process assets</u>. Encoding, formalization, or automation of a work process in the domain, typically used as a tool, decision aid, etc. rather than a reusable component to be directly incorporated into systems. Potential process assets mainly emerge in the *Interpret Domain Model* task within the *Refine Domain Model* sub-phase of *Model Domain*. In the *Select Features and Customers* task the surrounding context and rationale for these process assets is investigated. Often, the result can be discovery of potential types of assets never envisioned at the start of the domain engineering project.

- <u>Stakeholder-based assets</u>. Closely related to process assets, stakeholder-based assets are structured around the roles, profiles, or experience of particular stakeholders. Since a given stakeholder may utilize multiple workproducts and perform many processes, opportunities for encapsulating reusable knowledge into assets may emerge independently of artifact or process structure. For example, automated or codified expert knowledge could be used in decision making or workproduct transformation. Use of products of domain engineering (like the DOMAIN MODEL itself) by domain practitioners will often take the form of stakeholder-based assets.

➤ Allocate selected features to domain contexts

Allocate features to each DOMAIN CONTEXT included within the scope of the ASSET BASE. A feature that affects the implementation language for a component would be allocated to the System in Development context. A feature that specifies the behavior of a component is allocated to the System in Operation context.

➤ Contract with customers for commitment

This is in large part a validation step, but an essential one at this point. The asset base is going to be designed based on a set of assumed customers. Before proceeding further, some contracting with these customers must be done.

## Workproducts

■ ASSET BASE MODEL

- ASSET BASE FEATURE MODEL: The set of features to be supported by the ASSET BASE.
- ASSET BASE CUSTOMER MODEL: Asset utilizers (customers) characterized by attributes of their context of application, profile (e.g., experience level, role in various life cycle phases, etc.).
- ASSET BASE FEATURE — CUSTOMER MAP: The mapping of selected features to selected customers. This is a refinement of the PRIORITIZED FEATURE — CUSTOMER MAP produced in the *Prioritize Features and Customers* task.

## Guidelines

- Consider resource trade-offs. The final model can be optimized in at least three ways:
  - — Satisfy core customers with the minimal set of features that will obtain their commitment as users of the asset base; beyond this point, use minimum project resources;
  - — For the allocated resources, satisfy the core customers with the richest set of features of interest to them;
  - — For the allocated resources, satisfy the broadest market of customers possible.
- Consider trade-offs between optimization and generality. One primary trade-off will be between closeness of fit to particular exemplars (at this point, customer contexts) versus the flexibility to be used by multiple customers. A priority to integrate with one existing customer system will create strong pressure to adopt an architectural approach compatible with that exemplar system. To avoid undue influence of one existing architecture make sure you have studied several as diverse as called for by the domain definition.

### Exit Criteria

- Core customers' needs have been addressed in the prescriptive features selected for the ASSET BASE MODEL.
- The features selected for the ASSET BASE correspond to one of the layers identified in the EXTENDED DOMAIN MODEL. This ensures that the features supported will not have gaps or excess coverage that are responsive only to the selected customers.
- The features selected in the ASSET BASE FEATURE MODEL are feasible given the project constraints.

### Validation and Verification

- Since definition of the ASSET BASE MODEL involves both top-down and bottom-up processes, some integration and cross-validation will be required, particularly if the activities have been performed in parallel. Specifications of individual feature variants, feature configurations and architectural variants to be supported are integrated and checked for both internal and cross-consistency.

## 7.2 Architect Asset Base

### Purpose

The *Scope Asset Base* sub-phase has produced an ASSET BASE MODEL that specifies the range of features to be supported by the ASSET BASE and the targeted customers for the assets. This model even captures to some extent which *combinations* of features would be desired within a particular application contexts, e.g., by specifying the desired feature binding sites for selected features.

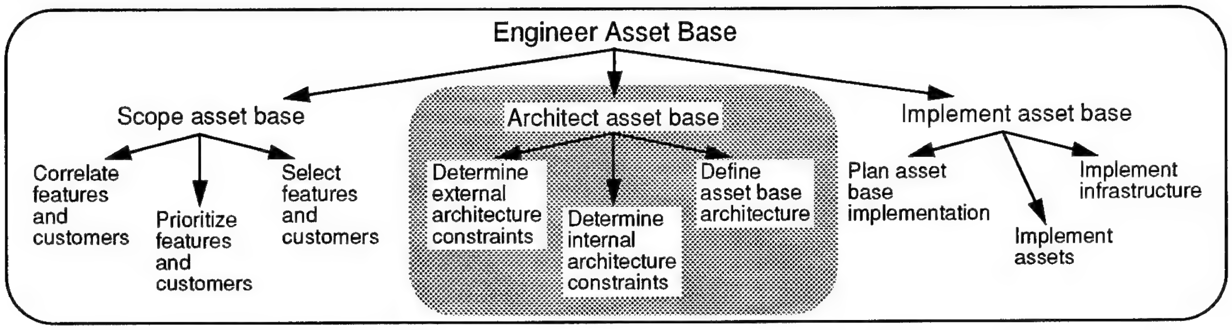

Engineer Asset Base
Scope asset base
Architect asset base
Implement asset base
Correlate features and customers
Prioritize features and customers
Select features and customers
Determine external architecture constraints
Determine internal architecture constraints
Define asset base architecture
Plan asset base implementation
Implement assets
Implement infrastructure

**Exhibit 61.** Architect Asset Base Process Tree

However, as yet the selected features have not been allocated to specifications for individual assets. There are many potential axes of variability for implementing assets that support these feature profiles, including:

— Whether desired functionality would be encapsulated for customers in monolithic components or aggregates of smaller components;

— How much internal reuse will occur within assets, i.e., how strongly interconnected the asset base as a whole will be;

— Whether alternative functionality required by different customers will be implemented as separate variants, or generated/transformed as required.

The primary purpose of the *Architect Asset Base* sub-phase is to isolate a subset of these design decisions at the level of the ASSET BASE ARCHITECTURE. Architectural concerns include ascertaining to what extent the design and implementation of assets is constrained by the external interfaces anticipated in different customer contexts, as well as determining the internal structure and interconnections of components in the asset base. The result of this sub-phase is the ASSET BASE ARCHITECTURE and a set of ASSET SPECIFICATIONS. Each of the latter specifications is subject to an independent technology selection process in the next sub-phase, *Implement Asset Base.*

Separating the asset base engineering process into these sub-phases addresses the following objectives:

— Localize the impact of shifts or evolution in implementation technology; e.g., conversion of an individual family of static components to a generator. This facilitates later migration to alternative implementation strategies with minimal disruption to overall asset base structure.

— Defers consideration of opportunities to reengineer legacy components until *after* a coherent architectural approach is determined. Facilitating this reengineering is a goal of the ODM process, and significant effort goes into preserving information that will support it. But, in principle, such reengineering actually re-introduces an element of ad hoc or opportunistic reuse into the heart of the *Engineer Asset Base* phase of domain engineering. Allowing design decisions to be guided primarily by these concerns would risk undoing much of the value derived from earlier steps of the process. The process recommended here embodies a sequencing strategy designed to mitigate this risk.

— Manages potentially explosive variability. Constraints on variability in the feature space have been introduced throughout the domain engineering life cycle; nevertheless, two important frontiers of variability need to be addressed at the architectural level at this point in the process.

External interface constraints from multiple systems can create requirements for variability that cannot be met in a cost-effective way; i.e., building two separate variants to address two separate contexts of use may achieve no net economy of effort.

In addition, the fact that subsets of assets may be used as ensembles in different customer contexts introduces an order of magnitude increase in the number of potential variant structures to be considered. This is not the case in engineering for a single system, where the structure has impact primarily on maintainability and related engineering factors.

## Description

As shown in Exhibit 62, the *Architect Asset Base* sub-phase consists of three major tasks:

- The *Define External Architecture Constraints* task addresses external interfaces to asset functionality required by each of the selected customer contexts specified in the ASSET BASE MODEL. These include constraints imposed by the external system environments in which assets or ensembles of assets will operate, as well as external services that may be invoked by

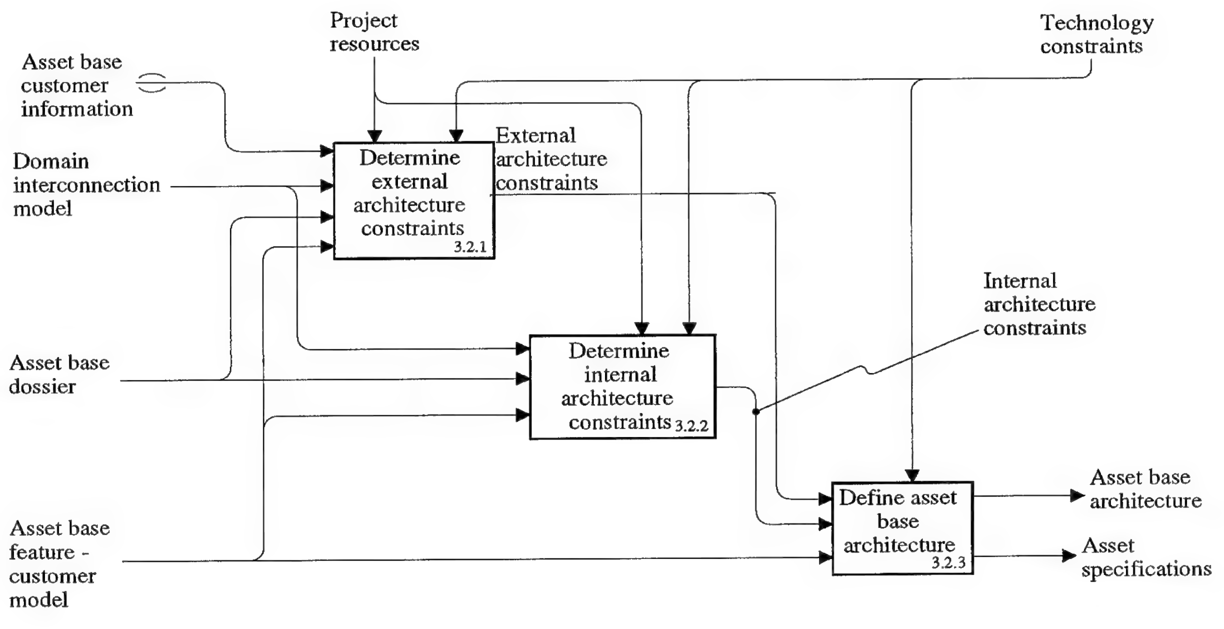

Project resources
Technology constraints
Asset base customer information
Domain interconnection model
Determine external architecture constraints 3.2.1
External architecture constraints
Internal architecture constraints
Asset base dossier
Determine internal architecture constraints 3.2.2
Asset base feature - customer model
Define asset base architecture 3.2.3
Asset base architecture
Asset specifications

**Exhibit 62.** Architect Asset Base IDEF$_0$ Diagram

asset functions. In practice, there may be some overlap between these two kinds of external constraints, which necessitates their being dealt with in an integrated way. The common thread is that these are constraints over which asset base engineers have *no direct control* (other than to exclude consideration of the context in question by iteratively adjusting the ASSET BASE MODEL).

- The *Define Internal Architecture Constraints* task addresses the issues of how to structure the internal relationships between assets in the asset base as a whole, and in particular among assets that will be used in ensemble in particular customer contexts. In this task there are also two complementary perspectives considered: layering of assets in separately selectable subsets, each of which represents a restricted version of overall functionality; and ***component*** encapsulations of particular domain functions. Constraints on architecture are assessed with respect to these two axes of variability.
- The *Define Asset Base Architecture* task integrates the constraints identified from the first two tasks and performs the final trade-off analysis that results in the ASSET BASE ARCHITECTURE. Committed features are allocated to individual ASSET SPECIFICATIONS. Each specification represents a partitioned area of the asset base that may be implemented with separate components, generative techniques, or hybrid strategies. In the *Implement Asset Base* sub-phase, these specifications will be mapped to specific implementation strategies. The architecture also specifies high-level interactions among assets.

### Sequencing

- The *Determine External* and *Determine Internal Architecture Constraints* tasks can be performed sequentially, in parallel or iteratively.

  For a single-system engineering context, it would make most sense to determine external constraints first, and use these to filter the options for internal structuring decisions. While this sequence also makes intuitive sense in the domain engineering context, there is the added complexity that external constraints reflect an intended multi-system scope of applicability itself determined (to some extent "designed") by the asset base engineers. This means that in principle opportunities or constraints emerging from internal architectural issues could influence external decisions such as the range of external facilities to address or the specific interfacing mechanisms supported.
- Managing the potential circularity in this process is part of the rationale for separating both tasks from the *Define Asset Base Architecture,* where binding decisions are made. Even though there may be circular dependencies in the various constraints identified in the first two tasks, the activities of documenting these constraints can terminate without trying to resolve conflicts in the same pass. The final task in the sub-phase can then apply decision techniques at a more or less formal level to produce an acceptable set of architectural choices.

## 7.2.1 Determine External Architecture Constraints

### Purpose

By determining these constraints here, the ASSET BASE ARCHITECTURE can be tuned to those constraints relevant to the selected customers.

In an industry where domain engineering methods are mature and pervasive, many of the external interfaces identified would be interfaces to other asset bases or potential asset bases. However, for the foreseeable future domain engineering projects will be "islands of reuse" in an ocean of leg-

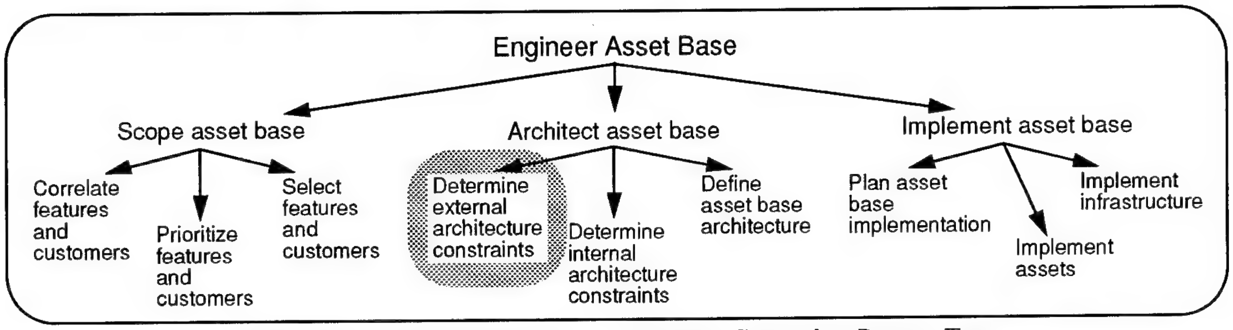

Engineer Asset Base
Scope asset base
Architect asset base
Implement asset base
Correlate features and customers
Prioritize features and customers
Select features and customers
Determine external architecture constraints
Determine internal architecture constraints
Define asset base architecture
Plan asset base implementation
Implement infrastructure
Implement assets

**Exhibit 63.** Determine External Architecture Constraints Process Tree

acy systems. So identifying the smallest range of variability at external interfaces is an important scoping technique.

> Example. In the Outlining domain, one key interface may be to a text editor that invokes the outline processor as a sub-system. The "Text Editor" domain will have been flagged as a related domain in the DOMAIN INTERCONNECTION MODEL, so that minimal effort has been expended on modeling variability that is dependent on the text editor chosen. To implement the asset base, though, engineers now need to determine the *specific* text editor systems the assets will need to accommodate.

## Entrance Criteria

- The ASSET BASE MODEL has been specified in the *Scope Asset Base* sub-phase.
- Information about selected customer contexts has been documented in the CUSTOMER CONTEXT DOSSIER.

## Inputs

- DOMAIN INTERCONNECTION MODEL. Structurally related domains reveal key interfaces with trade-offs such as strong vs. weak coupling
- ASSET BASE MODEL. Source of feature sets to be implemented and associated customer contexts. The PRIORITIZED FEATURE — CUSTOMER MAP provides information for determining the constraints relevant in each customer context.
- ASSET BASE DOSSIER. Information about representative systems studied in the *Model Domain* phase and customer contexts studied in the *Scope Asset Base* sub-phase.
- ASSET BASE CUSTOMER INFORMATION: A component data flow of CANDIDATE CUSTOMER INFORMATION, this includes new information elicited from selected customers about architectural constraints of systems targeted for implementing or migrating domain assets.

## Controls

- PROJECT RESOURCES: A component data flow of PROJECT CONSTRAINTS; these include both constraints on resources such as budget and schedule, as well as qualification of specific resources available for particular architectural options.
- TECHNOLOGY CONSTRAINTS: A component data flow of PROJECT CONSTRAINTS; these include any restrictions on strategies for architecting the asset base. These are distinct from the architectural constraints to be elicited from ASSET BASE CUSTOMER INFORMATION which concern

constraints on system architectures in which assets will be integrated. The constraints meant here directly affect the architectural choices of the asset base implementors.

Example. In the Outlining domain, an example of an external architecture constraint might be a given customer's requirement that the outliner interface with a particular word processor program, which has its own internal architecture and protocol for communication with other programs. In addition, because the project is in part a technology evaluation/demonstration project, the outline assets must be implemented using an in-house generative tool—an example of a technology constraint on the project.

## Activities

➤ Identify key external interfaces

Starting from the operational and application contexts documented in the DOMAIN INTERCONNECTION MODEL, identify key external interfaces to the asset base functionality to be implemented. Document these in the KEY EXTERNAL INTERFACES list.

Note that what is defined as an external interface at this point in the life cycle may differ from the definition during the *Model Domain* phase. A related area of functionality is now considered external if it is outside the scope of what will be implemented for the asset base. Since the ASSET BASE MODEL is a subset of the DOMAIN MODEL, some functional areas within the DOMAIN MODEL scope will appear as external interfaces to the ASSET BASE MODEL. Other areas within the DOMAIN MODEL scope may not need to be modeled as interfaces at all.

*Example.* In the Outlining domain, suppose that the DOMAIN MODEL includes the functional area of generating outline-style numerals for outline documents as one feature area. However, in the *Scope Asset Base* sub-phase it is determined that the primary customers targeted for the initial release of the asset base are users who rarely use this feature. When required it often calls for sophisticated typographic processing that is deemed outside the core functional area of the domain. This component would now be treated as an *external* interface. The external architectural problem now becomes finding a way to allow applications builders to code their own routine of this kind and link it if needed with the core outlining capability.

On the other hand, if the ASSET BASE MODEL excludes outlines dealing with graphical displays of outline structure, then an interface with graphic display routines may not be required as part of the ASSET BASE ARCHITECTURE.

➤ Partition prescribed features to key external interfaces

Determine the features within the ASSET BASE FEATURE MODEL that constrain each interface, and allocate the features accordingly. This refines the allocation of features to contexts that was done in the ASSET BASE MODEL to a greater level of detail in both features (e.g., individual features and feature variants within the feature set) and interfaces relevant to each context. Document the results in the FEATURE — EXTERNAL INTERFACE MAP. A template of this workproduct is illustrated in Exhibit C-14.

➤ Determine constraints of customer contexts on each interface

For each customer in the ASSET BASE CUSTOMER MODEL (part of the ASSET BASE MODEL) document constraints on each external interface. Architectural constraints of legacy systems that are potential contexts for asset base utilization as well as requirements for new systems. Much of this information should be available in the CUSTOMER CONTEXT DOSSIER. Supplemental ASSET BASE

CUSTOMER INFORMATION is obtained through direct contact with customers. Document the results in the CUSTOMER — EXTERNAL INTERFACE MAP. A template of this workproduct is illustrated in Exhibit C-12.

> *Example.* In the Outlining domain, for a stand-alone outlining utility that requires access to multiple text editing systems, a loosely-coupled interface might be most efficient. For an outliner embedded in a larger word processor, seamless and tightly integrated use of the host editor might be required. To build a set of assets that can be used to configure *both* versions, several variants might need to be implemented and separately supported.

## Workproducts

■ EXTERNAL ARCHITECTURE CONSTRAINTS

- KEY EXTERNAL INTERFACES. A list of the interfaces to ASSET BASE functionality to be supported, for each domain context for which reusable ASSETS are to be developed. Interfaces include environmental and sub-system interfaces. Environmental and sub-system interfaces can be partitioned if appropriate, but it is possible that the determination of which category an interface falls into may depend on final architectural decisions that will be decided in a later task.
- FEATURE — EXTERNAL INTERFACE MAP. Allocation of features from the ASSET BASE FEATURE MODEL to the interfaces identified in each customer context. A template of this workproduct is illustrated in Exhibit C-14.
- CUSTOMER — EXTERNAL INTERFACE MAP. A matrix documenting relevant constraints for each customer in the ASSET BASE MODEL has on the form of each external interface. A template of this workproduct is illustrated in Exhibit C-12.

## Guidelines

- Consider interface coupling trade-offs. Most guidelines for designing reusable software advise developers to "minimize context dependencies" or encapsulate components as a general design rule. But design for reuse is harder than conventional system design in large part because each of these design decisions involve trade-offs, the impact of which must be assessed for multiple application contexts.

  Where possible, encapsulate asset base functionality for easy integration with other software resources. Treat calls from domain services to external sub-functions as cleanly as calls from the external system environment to asset services. This will allow several kinds of "plug and play" flexibility when integrating asset base functionality into applications:

  — Depending upon the stability of the interface, asset base functionality could be tailored by integrating with different external components; and/or

  — Asset base functionality could be treated as a component that could be swapped, for example, with other implementations based on the same initial DOMAIN MODEL.

  However, in practice not all interfaces will be equally or simultaneously amenable to clean encapsulation. This will depend greatly on the relative maturity of the related domain. Graphical user interfaces (GUIs) have now evolved to the point where applications can be written with clean layers to the GUI; many other relatively "horizontal" functional areas have not evolved to this degree. In addition, there are interactions and constraints among the various interfaces.

### Exit Criteria

- All key external interfaces have been identified, and the list has been filtered according to relevant constraints.

  This ensures that all the parameters to be considered in the trade-off analysis have been collected, but that no analysis on easily excluded options will be performed.

- All relevant constraints have been documented for each customer.

  There has been no attempt to screen out certain sets of constraints to achieve compatibility. In general the set of constraints identified will *not* be mutually compatible.

### Validation and Verification

- Formally validate the documented constraints for *completeness* relative to the structural relations in the DOMAIN INTERCONNECTION MODEL.
- Validate for *consistency*: where the representations used force redundancy, make sure the data is consistent, particularly if different modelers have performed activities separately. E.g., if two customer contexts impose the same architectural constraint, the estimate of effort required to meet that constraint should be the same in each case.
- Validate by obtaining feedback from representatives among customers. Make sure that information in the ASSET BASE DOSSIER that may have been carried over from the DOMAIN DOSSIEr is still current and accurate.

## 7.2.2 Determine Internal Architecture Constraints

### Purpose

The ASSET BASE MODEL specifies the *overall* set of features that will be supported by the ASSET BASE, as well as a specific set of customer contexts that will be targeted as utilizers of the assets. The mapping between them establishes which overall profile of features will be desired by each customer. In a single-system engineering context, the next step would be to allocate the required features to components of an overall system architecture. In the domain engineering context, there are several other dimensions of variability to be considered. In this task, engineers determine constraints on the *internal* structural variability of the asset base. This is complementary to the *external* constraints determined in the previous task. The tasks can be done sequentially or iteratively.

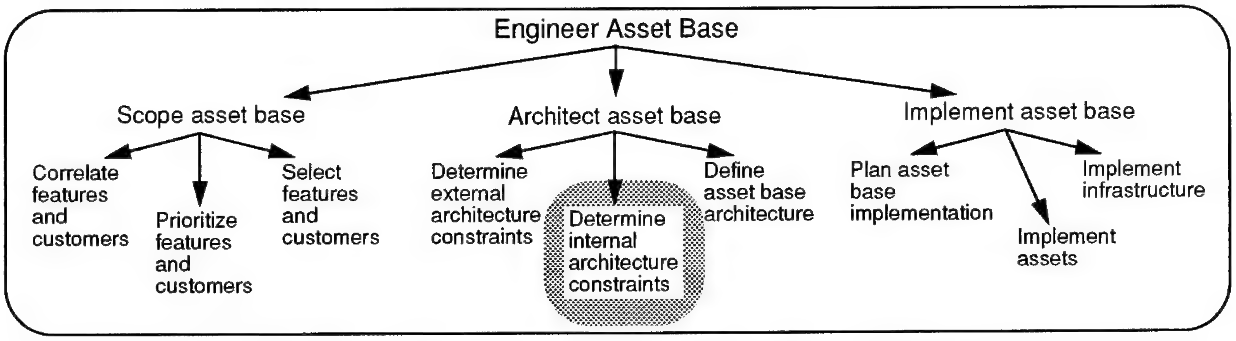

Engineer Asset Base
Scope asset base
Architect asset base
Implement asset base
Correlate features and customers
Prioritize features and customers
Select features and customers
Determine external architecture constraints
Determine internal architecture constraints
Define asset base architecture
Plan asset base implementation
Implement assets
Implement infrastructure

**Exhibit 64.** Determine Internal Architecture Constraints Process Tree

The primary purpose of performing this task is to ensure that all constraints have been identified that could affect the predictability of different combinations of assets in independent use. A secondary benefit of the task is to identify opportunities for achieving greater levels of usability for different customers by providing useful subsets of functionality that can be implemented and maintained in a cost-effective manner.

These subsets of functionality can take two primary forms in the domain engineering context. Sets of sub-components that form restricted versions of the overall domain functionality are called ***specializations***. Other subsets of functionality are referred to as ***components*** or ***subsystems*** within the scope of the asset base. There may be complex interactions affecting the separate selectability of layers or components; a primary function of this task is to document these INTERNAL ARCHITECTURE CONSTRAINTS.

## Entrance Criteria

- The ASSET BASE MODEL has been specified in the *Scope Asset Base* sub-phase.
- Technology constraints that could reduce the set of structural variants considered have been identified.

Determination of the EXTERNAL ARCHITECTURE CONSTRAINTS need not be complete to begin this task.

## Inputs

- DOMAIN INTERCONNECTION MODEL. Source for component, specialization and specialization subdomains that determine key *internal* interface issues for the asset base.
- ASSET BASE MODEL. Source for the ***feature ensembles*** required by various customers for the asset base. These represent the complete ensemble of functionality in the domain that must be available to given customers. The feature ensembles are allocated to specific domain contexts (as for external constraints).
- ASSET BASE DOSSIER. Important if developers want to consider leveraging legacy artifacts as a criterion in selecting architecture

## Controls

The following controls are close parallels to those for the previous task, *Determine External Architecture Constraints.*

- PROJECT RESOURCES: A component data flow of PROJECT CONSTRAINTS; these include both constraints on resources such as budget and schedule, as well as qualification of specific resources available for particular architectural options.
- TECHNOLOGY CONSTRAINTS: A component data flow of PROJECT CONSTRAINTS; these include any restrictions on strategies for architecting the asset base.

  *Example*. In the Outlining domain, an example of an internal architecture constraint might be a given customer's requirement that the outliner function in a read-write as well as a browse-only mode. This involves selection of more than individual feature variants because the implications pervade most of the functions in the application. In addition, some developers who would incorporate the outline module would need to fix this mode at development time; others would offer it as a start-up option to the end-user. These are internal constraints because they affect the topology, structure, interconnections and binding of aggregates of components developed in the asset base.

Technology constraints would also include limitations in technology that affect options for supporting variability in architectures.

## Activities

➤ Factor feature sets into feature ensembles

Map the FEATURE SETS, allocated to different customers and contexts as documented in the ASSET BASE MODEL, to FEATURE ENSEMBLES. Each ensemble encompasses a set of features that needs to be accessed as a whole by practitioners within a given context. Each ensemble selected for support will be implemented by some mix of assets using component or generative techniques. Record the results in a FEATURE ENSEMBLE — CUSTOMER CONTEXT MAP.

The overall FEATURE SET associated with a given customer context becomes its own FEATURE ENSEMBLE by default. Other ensembles are identified by *decomposing* these overall ensembles into subsets. The result can be a different set than that identified in the ASSET BASE MODEL since that task did not systematically consider *all* subsets of selected feature sets. Note that only those subsets identified as part of the DOMAIN MODEL are considered. (If new subsets are discovered or proposed, they ideally should be flowed back through the *Extend Domain Model* task in the Model Domain phase.)

*Example*. For a system-in-development context, there may be design features to provide code components in either functional or subroutine form. If a typical user would call only one form of component, these two features could be partitioned separately. If the same engineer would need access to both forms of component, these two features become effectively part of one FEATURE ENSEMBLE.

➤ Determine component subsystems

***Separate selectability*** refers to accommodation of variation by encapsulated sub-components of an architectural framework that can be included or deleted from system instances relatively independently. For any system S with subsystem S1, consider the following variant instantiations:

— The sub-component as a stand-alone variant (S1);

— The system with the sub-component ***masked*** (S - S1);

— The system with the sub-component (S, including S1).

For each component subsystem in the asset base, consider the following architectural issues:

- Can the component be masked? I.e., can a system variant be produced without the component's functionality included?
- Can the interface to the component be encapsulated to allow for alternative implementations of the component?

*Example*. In the Outlining domain, could a sub-system to generate heading level numbering be masked out of a system version where this function would not be required? Could the program be parameterized in such a way that user-written heading numbering routines could be provided?

➤ Determine specialization/layered subsystems

Consider ensembles of features that implement the full functionality covered in the domain at varying degrees or levels. These can be considered feature ensembles that will correspond to specialization or layered subsystems of asset base functionality. Document the constraints on each ensemble of this type. The documented constraints include component and layering constraints. A template of a COMPONENT CONSTRAINTS MAP workproduct is illustrated in Exhibit C-9. A template of a LAYERING CONSTRAINTS MAP workproduct is illustrated in Exhibit C-10.

It may be helpful to document the semantic relations between specializations using a *layered architecture* approach for the feature sets contained within the scope of the ASSET BASE FEATURE MODEL. In a layered approach, an inner kernel with minimum or core functionality is surrounded by successive layers, each of which provides enhanced functionality in some respect. For each candidate architectural layer, consider the following issues:

- Will the layer be separately selectable as an instantiation of domain functionality? (See the discussion on separate selectability in the Guidelines section below.)
- If instantiated as a separately selectable variant, will the layer need to be *optimized* with respect to its own internal layers?
- If incorporated into outer layers, will there be separate interfaces to the layer? For example, most interactive programs also provide application program interfaces (APIs). An asset base could be architected so that there are several potential API levels available; not all will be desired in every instantiation.

➤ Consider internal coupling trade-offs

Consider trade-offs of strong vs. loose coupling of assets within the structure of the asset base. Strong coupling will increase the internal reuse and modularity of assets, but will create more asset-to-asset interdependencies. Loose coupling will preserve greater autonomy of separate assets, but usually with additional overhead. The burden may fall as much on the configuration management and version control mechanisms available as on the implementations themselves.

While cohesion vs. coupling trade-offs are familiar from general software engineering, they present special problems here because each asset may be required to perform within a number of different instantiations of domain functionality. These requirements will impose significant constraints on the ASSET BASE ARCHITECTURE.

Some of these constraints and trade-offs can be derived from semantic information about the features mapped to each ensemble. Since the asset base architecture defined at this stage links only the asset ensemble specifications as a whole, it can record only the general fact of this dependency between the two asset ensembles.

> *Example.* Suppose one feature ensemble represents a family of variant algorithms used within the domain of focus, while another ensemble represents variants of a key data structure used in the algorithm calculation. For any instantiation of the algorithm assets there may be restrictions on what variants of the data structure asset(s) can be instantiated as part of the same system.

## Workproducts

■ INTERNAL ARCHITECTURE CONSTRAINTS

- FEATURE ENSEMBLE — CUSTOMER CONTEXT MAP. Documents which feature ensembles are *desirable* in contexts of use and which feature ensembles are *feasible* in contexts of development. Refines the information in the ASSET BASE MODEL.
- COMPONENT CONSTRAINTS MAP. A map of the feature ensembles that can potentially be selected as stand-alone or separately selectable. A template of a COMPONENT CONSTRAINTS MAP workproduct is illustrated in Exhibit C-9.
- LAYERING CONSTRAINTS MAP. A map of subsets/layered architecture variants to feature ensembles. A template of a LAYERING CONSTRAINTS MAP workproduct is illustrated in Exhibit C-10.
- FEATURE ENSEMBLE CONSTRAINTS. Separate selectability constraints and other interactions among ensembles.

## Guidelines

- Select a small set of combinations for analysis. Separate selectability makes the architectural problem inherently more complex for reuse than in the design of a single system. The *same* component may appear as an element of several overlapping, separately selectable subsystems obtainable from the overall architecture. In the most complex case, this could lead to a combinatorial explosion of architectural variants for a given component topology.

  *Example.* Few compilers are architected so that the lexical analyzer, parser, and/or code optimizer are all easily accessible as separate utilities. In engineering a single system, each component can be designed to be modular, perhaps even separately selectable on an individual basis. (This is rarely applied systematically; for example,).

  Often in domain engineering no one way of partitioning the system into components will meet the needs of multiple potential contexts of use. Modelers need to select a small subset of these combinations to support as separately selectable for any practical asset base.

- Components and specializations are dual perspectives. The distinction between components and specializations of the asset base functionality can become blurry in cases where a large proportion of functions are provided. That is, some ensembles can be difficult to classify clearly in one category or another. A system with one component masked out may be classified as a specialization or restricted-functionality variant of the system.

  Do not spend much effort in looking for the "right" allocation. As long as each feature ensemble has been covered in the scope of at least one activity, the allocation is not critical to the main purposes of the task.

## Exit Criteria

- The feature ensembles defined in the ASSET BASE MODEL have been allocated to internal architectural categories.
- The trade-offs for each ensemble in terms of masking, separate selectability, and optimization of internal structure have been documented.

The final selection of encapsulation, optimization and separate selectability attributes to support are determined in the subsequent task, *Define Asset Base Architecture.*

### Validation and Verification

- Validate that each ensemble has been allocated only once (see Guidelines) as a component or layer.
- Validate the trade-off analysis for internal consistency.
- Validate assumptions of usability/feasibility for individual ensembles with asset base customers.

## 7.2.3 Define Asset Base Architecture

### Purpose

The two previous tasks in this sub-phase *Determine External* and *Internal Architecture Constraints* respectively for the ASSET BASE to be developed. The primary purpose of the task, and the rationale for performing it separately from and subsequent to the earlier tasks, is to enable this decisions to be made by integrating both internal and external constraints simultaneously and in a managed way. The benefit is that higher-quality architectural strategies will be revealed that may not have been chosen via more informal methods. In particular, this strategy is oriented towards managing the potential complexity of highly variable architectures, which makes the process robust enough for a wide variety of domains.

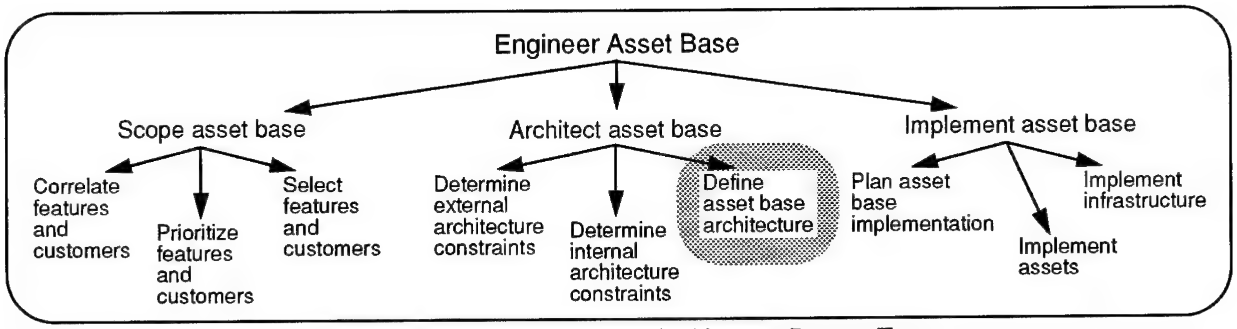

Engineer Asset Base
Scope asset base
Architect asset base
Implement asset base
Correlate features and customers
Prioritize features and customers
Select features and customers
Determine external architecture constraints
Determine internal architecture constraints
Define asset base architecture
Plan asset base implementation
Implement assets
Implement infrastructure

**Exhibit 65.** Define Asset Base Architecture Process Tree

In this task the final selection is made for a range of external and internal architectural choices:

— the range of external interfaces to be supported;

— the degree of coupling and other interconnection strategies to be supported for each interface;

— which components within the scope of the asset base will be implemented to be separately selectable, masked, optimized, etc.

These results are documented in the ASSET BASE ARCHITECTURE.

## Entrance Criteria

- The ASSET BASE MODEL should be complete. The trade-off analysis performed in this task attempts to optimize for the full set of features and customers to be supported. The task activities would need to be repeated if these change significantly.
- Initial results of the previous two tasks should be available. At a minimum, enough constraints should have been documented to highlight the trade-off issues involved.

## Inputs

- EXTERNAL ARCHITECTURE CONSTRAINTS. Key input to the architecture trade-off analysis and selection of variants to be supported.
- INTERNAL ARCHITECTURE CONSTRAINTS. Key input to the architecture trade-off analysis and selection of variants to be supported.
- ASSET BASE MODEL. ASSET BASE CUSTOMER MODEL provides input for prioritizing the variants to be selected.

## Controls

- TECHNOLOGY CONSTRAINTS.

Project resources are *not* a control on asset specifications. Development of asset specifications do not constitute a commitment to develop all assets specified within the scope of the domain engineering project.

## Activities

➤ Map internal to external architecture constraints

➤ Consider alternative architecture approaches

Consider alternative architectural approaches to accommodate the specific feature and (system) architectural variants committed to for the ASSET BASE MODEL.

Bearing in mind that the ASSET BASE ARCHITECTURE is the focus of this design activity, there are several architectural approaches that are worth particular attention as strategies for accommodating variability. Each can be seen as a response to specific challenges of variable architectures.

- A component or subsystem within an asset base architecture may be configured to be part of multiple asset ensembles.

➤ Consider variability requirements for system architecture partitions

Assume that the features in the ASSET BASE MODEL have been allocated to initial ASSET BASE ENSEMBLES.

➤ Develop unified asset base architecture

After performing trade-off analysis on the architectural alternatives considered, select and document the best alternative. The resulting ASSET BASE ARCHITECTURE structures the set of asset ensemble specifications that make up the functional scope for the asset base.

➤ Allocate features to architectural components

This can be done in several steps. Since features were previously mapped to partitions, first map these partitions to architectural components, then derive the feature-to-architecture mapping.

➤ Transform allocated features to asset specifications

➤ Develop utilization profile for each asset specification

## Workproducts

■ ASSET BASE ARCHITECTURE

The ASSET BASE ARCHITECTURE models the range of variability in system architectures to be obtainable from the ASSET BASE. The architecture includes separately selectable asset ensembles. It should be as implementation-neutral as possible.

■ ASSET SPECIFICATIONS

ASSET SPECIFICATIONS contain specifications for assets that implement the selected features. The ASSET SPECIFICATIONS include the required range of variability in the ASSETS, and constraints and interdependencies with other ASSETS.

## Guidelines

- Defer implementation choices where possible. Allow for different assets to be implemented with different technology choices (e.g., generative, constructive). This will support heterogenous domains where one across-the-board technology approach is not suitable. Local application of technology can support feasibility in some cases. e.g., smaller generative tasks are more tractable.

  This strategy will also support evolution over time. Individual components may evolve toward generative implementations with repeated usage and refinement. The architecture should support this evolution as seamlessly as possible. Strive for independence with respect to evolving standards and conventions.

## Exit Criteria

- ASSET BASE ARCHITECTURE and ASSET SPECIFICATIONS have been defined that are implementable and that address the needs identified within the targeted customer contexts.

## Validation and Verification

- Formal validation. It will be difficult to test asset base architectures independently of the assets that it supports. Guidelines for formally evaluating architectures are still emerging.

Criteria might include: completeness of the architecture with respect to the targeted solution space, as represented by the subset of domain exemplar systems the architecture is committed to cover; consistency of the architecture models (both with themselves, and in relation to other models of the domain); and parsimony (e.g., do models allow specification of assets not intended for support?)

- Customer review. Allow potential customers to review the final asset base architecture, particularly from the standpoint of the ASSET ENSEMBLE specifications. It may be helpful to have a prototype of a potential utilizer's interface to the asset base as a way of getting this feedback. Customers would be, in effect, browsing the specifications only, so they won't be able to validate the assets themselves.

## 7.3 Implement Asset Base

### Purpose

The previous sub-phases in *Engineer Asset Base* have produced an ASSET BASE MODEL, ASSET BASE ARCHITECTURE, and ASSET SPECIFICATIONS; these workproducts, taken together, constitute the equivalent of requirements, architecture and high-level design for the assets in the asset base. The primary purpose of the *Implement Asset Base sub-phase* is to implement that portion of the specified asset base that is essential in order to transition it into use.

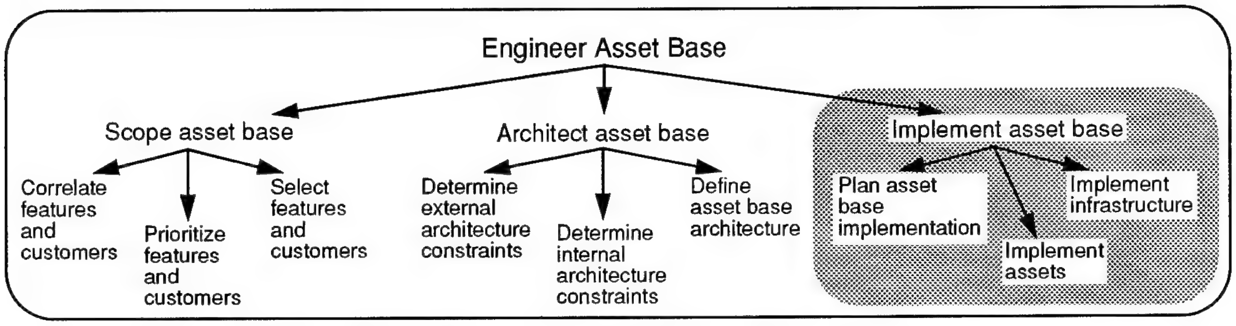

Engineer Asset Base
Scope asset base
Architect asset base
Implement asset base
Correlate features and customers
Prioritize features and customers
Select features and customers
Determine external architecture constraints
Determine internal architecture constraints
Define asset base architecture
Plan asset base implementation
Implement assets
Implement infrastructure

**Exhibit 66.** Implement Asset Base Process Tree

### Description

As shown in Exhibit 67, there are three major tasks in *Implement Asset Base*:

- The *Plan Asset Base Implementation* task involves selecting the technology to apply for each asset, planning implementation phases, and planning the test and validation strategy for the assets.
- The *Implement Assets* task carries out the core work of implementing the assets, in accord with the ASSET SPECIFICATIONS, and supported by the ASSET BASE INFRASTRUCTURE.
- The *Implement Asset Base Infrastructure* task implements support mechanisms for asset base customers, integrates the assets with general infrastructure mechanisms, and addresses any other asset base-wide implementation tasks.

### Sequencing

- Basic Sequence. In the most basic sequence the *Plan Asset Base Implementation* task is completed before the other two tasks commence. It is helpful to plan the implementation *phasing* strategy in particular before extensive implementation of particular assets, because the phasing strategy may allow for deferring implementation of certain assets, or ways to leverage certain other assets by developing them early.
- Individual asset implementation. In some other respects, though, this sub-phase deals with processes that can occur in the development of each individual asset; e.g., technology selection choices are made somewhat independently for each asset. So the entire *Plan Asset Base Implementation* task does not need to be complete before the other tasks begin.
- Early prototyping of assets. Taking the last point one step further, there may be advantages to implementing at least a few assets early in the sub-phase to validate implementation approaches being followed in the plan as a whole.

## 7.3.1 Plan Asset Base Implementation

### Purpose

The primary purpose of *Plan Asset Base Implementation* is to select implementation strategies for individual assets in the context of the ASSET BASE ARCHITECTURE as a whole. Technology choices can be made throughout the ODM life cycle; for the most part, these are considered part of general infrastructure planning and tailoring activities, and hence not included in the core process model. However, late binding of asset implementation technology choices is a significant feature of the ODM method. Deferring technology selection into the *Engineer Asset Base* phase helps ensure that technology choices fit domain characteristics, not merely implementors' past experience and technical biases. By deferring technology choices for individual assets, to the degree possible, until after specification of the ASSET BASE ARCHITECTURE, finer granularity in technology choices can be obtained (in contrast to, for example, an architecture-level decision to use an across-the-board generative or component-library approach).

To whatever degree technology-independence in the ASSET BASE ARCHITECTURE has been achieved, planning asset implementation as a unified up-front task facilitates coordinated rather than piecemeal use of heterogenous technologies, by considering interactions of technology choices within the ASSET BASE as a whole. In addition, it allows for systematic consideration of alternative (e.g. constructive vs. generative) approaches on a *per-asset basis*.

Other major purposes of this task include the following:

- Maximize use of legacy artifacts. Assets may be developed using a combination of newly developed workproducts and adaptations of existing artifacts characterized during the

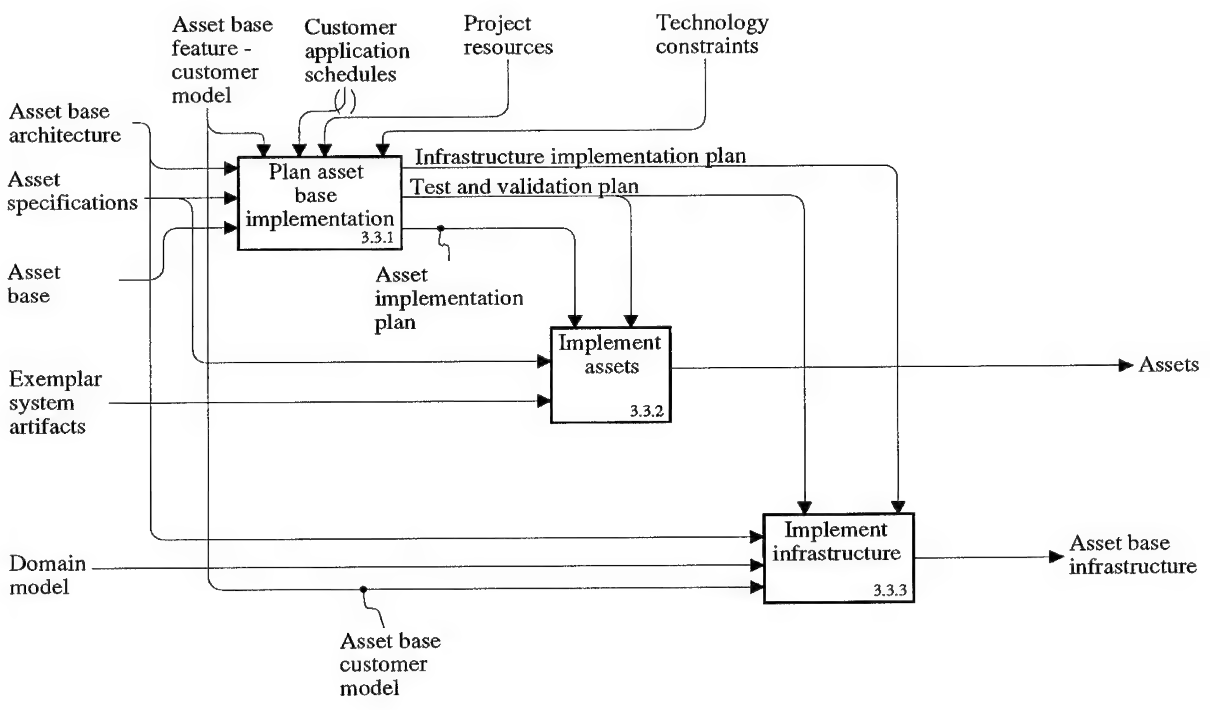

Asset base feature - customer model
Customer application schedules
Project resources
Technology constraints
Asset base architecture
Asset specifications
Asset base
Exemplar system artifacts
Domain model
Plan asset base implementation
3.3.1
Infrastructure implementation plan
Test and validation plan
Asset implementation plan
Implement assets
3.3.2
Assets
Implement infrastructure
3.3.3
Asset base infrastructure
Asset base customer model

**Exhibit 67.** Implement Asset Base IDEF$_0$ Diagram

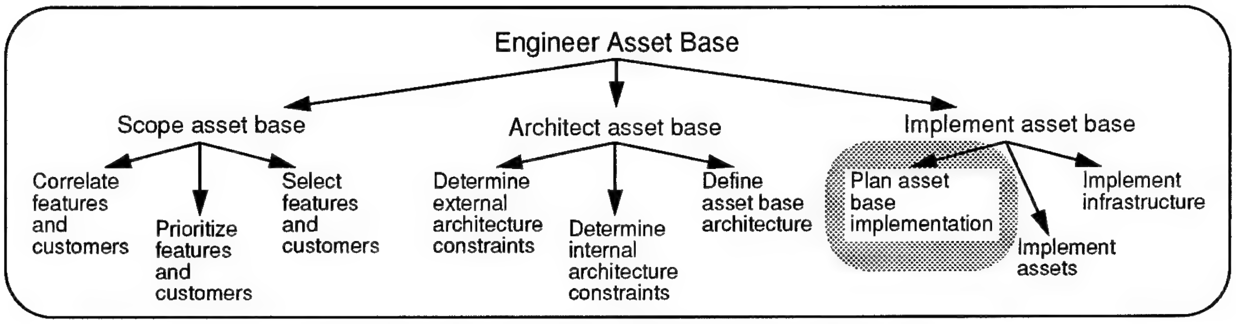

Engineer Asset Base
Scope asset base
Architect asset base
Implement asset base
Correlate features and customers
Prioritize features and customers
Select features and customers
Determine external architecture constraints
Determine internal architecture constraints
Define asset base architecture
Plan asset base implementation
Implement assets
Implement infrastructure

**Exhibit 68.** Plan Asset Base Implementation Process Tree

MODEL DOMAIN phase.

- Obtain economies in implementation. Determine the optimum strategy for incrementally implementing the assets and separately selectable subsystems; e.g., by preserving interim versions under separate configuration management so that the version is always obtainable from the asset base and reuse infrastructure.

- Factor application engineering project constraints. Select the best strategy for transitioning results of reuse engineering into application engineering groups at varying points in their project life cycles.

## Entrance Criteria

- Planning for asset base implementation should not commence until the individual asset specifications and the overall asset base architecture have been developed and validated.

  The primary advantage of the planning step involves making technology decisions in a coordinated way for the entire asset base, rather than on an asset-by-asset basis.

- Black-box testing strategy and even test plans could be derived for variants selected as early as completion of the ASSET BASE ARCHITECTURE and even, theoretically, the ASSET BASE MODEL.

  This approach would be similar to creating test plans directly from requirements or high-level designs in non-reuse-based software development. Such test plans might be fairly abstract, as they would need to be formulated independent of assumptions as to the ASSET BASE ARCHITECTURE or implementation technology; e.g., a test plan implementation would be quite different if the functionality to be tested were implemented as a family of components vs. a generator.

## Inputs

- ASSET SPECIFICATIONS. Feature profiles of individual assets to be implemented. A strategy is developed for each specified asset.

- ASSET BASE ARCHITECTURE. Overall architecture of the asset base. A strategy for developing the ASSET BASE INFRASTRUCTURE is derived from this workproduct.

- ASSET BASE DOSSIER. Contains usability and feasibility data, as well as traceability to possible prototype artifacts for reengineering.

## Controls

- ASSET BASE MODEL. Used to bound the features and customers and interface constraints considered.
- PROJECT RESOURCES. Needed to ascertain reasonable objectives and schedules for implementing assets.
- TECHNOLOGY CONSTRAINTS. Any global constraints (mandates, limitations) on technology that will affect the choices made per asset and for the asset base.
- CUSTOMER APPLICATION SCHEDULES. Used to develop the Project — Customer Time Line that helps determine the staging strategy for the asset base.

## Activities

➤ Identify customer constraints on technology selection

Potential customer contexts must be characterized more fully with respect to the actual implementation forms and packaging of the reusable assets to be provided. Some knowledge of the current forms of system workproducts (studied as artifacts) has been captured in the artifacts models during Descriptive Modeling. These models can now be reexamined and refined to express constraints on the implemented form for assets.

➤ Consider alternative asset implementation strategies

For each asset specification do a trade-off analysis, particularly on the benefits and risks of a component vs. a generative implementation approach. This trade-off analysis must consider several factors, including:

- The mechanisms for supporting variability in the implementation language (e.g., an object-oriented vs. functional language);
- The level of sophistication in configuration management that will be required in the asset base infrastructure;
- The potential for combinatorial explosion in feature combinations desired for the asset.

➤ Select Asset Implementation Technologies

One goal of previous sub-phases in the *Engineer Asset Base* phase of the ODM life cycle is to produce ASSET SPECIFICATIONS that reduce the *Implement Assets* task to a tractable problem for software engineering design techniques. Once this has been accomplished, the implementation technology applied is considered part of the Supporting Methods documented in Section 8.0. In general, the software engineering methodology used within the organization can be applied, with the following provisos:

- Adapt methodologies to reuse existing artifacts where possible.
- Adopt or develop standards for use across similar implementation tasks. E.g., the more separate generative tasks are required, the more important it is to have a standard meta-generative technology.

➤ Update ASSET BASE ARCHITECTURE and ASSET SPECIFICATIONS

Incorporate technology decisions into updated versions of both these sets of workproducts. The result can be considered technology-independent and technology-dependent versions respectively. Maintain these updated versions separately and in parallel with the original versions, if possible. This will facilitate possible roll-back later if technology decisions need to be considered, and will provide a "technology-hiding" specification layer that can be used in external reference specifications, so that flexibility is retained to evolve technology choices later.

➤ Consider customer context constraints on schedule

Consider the current life cycle phase for application projects that are potential utilizers of the assets to be developed. The overall plan should reflect opportunities and conflicts stemming from the schedule of each utilizer project. Adjust the schedule or phasing of implementation if necessary to coincide with these phases. This can make the difference between assets being used or being passed by, whatever their technical merits.

Record the results in a PROJECT — CUSTOMER TIME LINE report, as illustrated in Exhibit C-18. Place the anticipated project schedule with milestones for completed assets on a common time line with the anticipated schedules of potential customer applications. A separate phasing strategy may be required for each existing or anticipated system into which domain assets are to be incorporated.

> Example. One project might be in an early requirements definition and negotiation phase, where initial versions of domain assets could be used in a prototyping capacity. Another project may be almost through with detailed design, and therefore only be able to make use of thoroughly tested components that offer a close match to the functional profiles assumed by the current architecture. Each life cycle entry point presents different challenges.

➤ Plan Development Stages

Based on the PROJECT — CUSTOMER TIME LINE and the feasibility estimates for the various asset specifications, develop the detailed staging strategy for implementing assets. Consider factors such as requirements/opportunities for migrating/back-filling assets into existing systems; use of existing systems as testbeds/validation suites, and advantages of early release of certain sets of assets to facilitate adoption with particular customers. On a per-asset basis, the staging strategies can include any of the following options:

- *Prototype assets.* Assets implemented early enough to serve as validation of the DOMAIN MODEL, in particular the adequacy of the feature language as a specification of required functionality. Assets can also be done as prototypes in order to assess the technical feasibility of particular asset implementation techniques, methods and tools.

  None of these types of prototype assets are expected to be incorporated into the final ASSET BASE. The assumption is that final assets must be implemented in the context of an overall asset base architecture.

- *Assets to validate architecture.* A set of assets may be implemented that, taken as a whole, validate some aspect of the asset base architecture. For example, a set of routines may implement a given algorithm across a range of optimizations for diverse engineering factors. By implementing the entire set of these assets, the soundness of the technical approaches for supporting and demonstrating different optimization priorities can be validated. This still need not constitute implementation of the entire asset base.

- *Assets to validate infrastructure.* Similarly, a set of assets may be selected that, taken together, will allow for validation of key aspects of the asset base infrastructure (to the extent that these aspects are distinct from purely architectural factors).
- *Assets to validate utilization.* A set of assets adequate for incorporation into a complete system is probably highly desirable before attempting a full-scale implementation of all assets in the asset base. The key is that enough of the assets need to be implemented that system developers can experience the direct impact of having the assets available for use.
- *Assets implemented by asset utilizers.* Some assets may be included on an "implement as needed" basis. That is, the asset specifications (and possibly even the technical design, such as choice of programming language and encapsulation techniques) are prepared during domain engineering; but the actual implementation of the assets is left to the system developer when they are needed.

The ASSET IMPLEMENTATION PLAN should take into account decisions on implementation technology. In component-based engineering, the plan should specify the sequence of various builds or versions of domain assets to be developed. In generator-based engineering, the plan might detail plans for successively introducing greater levels of semantic completeness in the generator languages.

➤ <u>Plan Asset Base Testing and Validation</u>

Based on the types of assets to be implemented, the implementation techniques selected and the phased development plan, plan the strategy for testing and validation of both individual assets and the asset base as a whole. Document this in the TEST AND VALIDATION PLAN. The following paragraph highlight some points to consider.

*Test assets for conventional use*

Assets must be tested in a manner consistent with their conventional use within a domain application. There are stronger incentives to apply thorough testing techniques to assets that will be reused multiple times, in multiple applications. A reusable code component must be tested in analogous way, but more rigorously, than a component written for a single point of use; e.g., the component may need to be tested against several parallel sets of requirements, rather than one set of requirements.

The component may need to be tested more exhaustively, to satisfy a higher level of trust. It may need to be boundary-tested more thoroughly, so that its behavior when used in unanticipated contexts can be adequately predicted. Note that this is not equivalent to saying components must be maximally robust. Components can be engineered to make as strong or as weak assumptions about system-in-use contexts as is appropriate in the domain. The job of testing in an asset base engineering context is to verify that the component performs exactly as specified; i.e., testing for undocumented excess functionality as well as adequacy. It may also require system benchmarking tests at lower levels than would normally be required during system test, so that a library of variant components can be adequately characterized with regard to performance trade-offs.

*Additional requirements for asset testing*

Conventional software testing practices are oriented towards implementations (i.e., code). Asset bases can contain assets from across the software life cycle. The TEST AND VALIDATION PLAN should address how the various categories of reusable assets will be tested. Testing methods must be adapted to non-code artifacts that are to engineered for reuse to validate that they adequately support the intended range of variability. E.g., if design schemas are to be included in the asset base, some means of validating the design schemas with regard to their intended scope of applicability must be specified. The design component validation must ensure the design complies with relevant system constraints on form and structure.

In addition, assets may be developed using a spectrum of reuse engineering techniques, including component design, generative reuse, table-driven support for variability, macro expansion or hand-tailored templates. Each technique may require different techniques for testing and validation.

*Use model semantics in testing.*

Reusable assets must be tested, or qualified, with respect to their placement or classification within the ASSET BASE MODEL. A component that functions appropriately in a single system-in-use context may prove unacceptable when classified within the domain model because implicit contextual assumptions made in its implementation may violate conditions assumed by the model. Assets can fail by providing too much functionality, if their position in the asset base is intended to provide a specific set of features relative to other assets.

The TEST AND VALIDATION PLAN can take advantage of the defined semantic relationships between asset variants. Inheritance-based network modeling techniques allow relatively strong assumptions about the behavior of an asset based on its position within the network. Each category within the model can be associated with a set of possible asset implementations satisfying a particular set of requirements or conditions. Specializations within the network must satisfy the same set of requirements, plus possible additional constraints. Thus much of the testing protocol can be reused in multiple places within the model.

This will apply to overall system versions as well as specifications for individual assets. Because different versions or system configurations are characterized explicitly in terms of addition of features or relaxation or tightening of semantic restrictions, the TEST AND VALIDATION PLAN for the asset base can evaluate each more specialized version's compliance with new requirements and maintained compliance with old requirements.

NOTE: Not every test plan for an asset will be engineered as a "test plan asset". Such assets must be designed for a specified range of variability to be incorporated within the testing phase of different applications. (Of course, theoretically, such "testing assets", like all other assets, should be tested before inclusion in the asset base. This means that in principle test assets will require their own test plan!)

➤ Plan Asset Evolution

implementation of some asset specifications as a spin-off from system development

feedback from adaptation/integration of assets into applications

mechanisms for growing new components via aggregation, generation, user extension

## Workproducts

■ ASSET IMPLEMENTATION PLAN

Details of the technology to be used and the implementation schedule for each ASSET. Includes the following:

- PROJECT — CUSTOMER TIME LINE. A template of this workproduct is include in Exhibit C-18.

■ INFRASTRUCTURE IMPLEMENTATION PLAN

Contains a plan for the implementation of an ASSET BASE INFRASTRUCTURE. The INFRASTRUCTURE IMPLEMENTATION PLAN includes technology selections and scheduling constraints for infrastructure development relative to development of ASSET releases. The infrastructure may be specifically developed for the ASSET BASE or may be available technology that is applied to the ASSET BASE. In general, individual ASSET IMPLEMENTATION PLANS will be affected by the choice of ASSET BASE INFRASTRUCTURE.

■ TEST AND VALIDATION PLAN

## Guidelines

- Consider component sharing trade-offs. A primary design issue to be considered in this phase is the potential sharing of components among assets. Greater degrees of sharing will lead to a more strongly coupled asset base, which may minimize development effort and consolidate the workproducts produced. However, such strong coupling may work against the goal of separate selectability of subsystems and subsets of assets, as defined by the architectural variants to be supported.

- Consider outside-in vs. inside-out strategies. In domain engineering, a set of components that differ by progressively adding functionality (in terms of the features of the domain model) can be implemented via several alternative strategies. An incremental, or "inside-out" approach implements an initial system version with minimum functionality, then incrementally extends the functions with subsequent versions. maps each broader set of features to a later version, with new functions added sequentially.

  However, it can sometimes be more advantageous to implement the "richer" system version first, then to create subsequent versions that suppress, mask or strip away excess functionality to arrive at the "leaner" system variants required by the asset base architecture. One challenge in this approach is to make the functional reductions in such a way that the variants can be maintained as parallel versions. These complementary "inside-out" vs. "outside-in" alternatives are a recurrent trade-off issue throughout domain engineering life cycle. In typical single-system development, there is a much smaller "look-ahead window" for optimizing development in this way; hence the design trade-offs are less often confronted directly.

- Consider bootstrapping strategies for use of early components later in the development cycle. The ASSET BASE IMPLEMENTATION PLAN should capitalize on opportunities to build certain capabilities early in the implementation process that can be used later as well as becoming part of the final ASSET BASE. Besides achieving economies in development, this helps to validate tools and assets through direct and immediate usage.

- Consider scaffolding layers. A layered architecture may have been adopted for the overall asset base; e.g., a set of base components and a layer of composite functions available above this level. In implementation, it is sometimes desirable to specify additional layers at a finer degree of granularity than those layers intended for separate, persistent and/or independent access. Some additional layers may serve as scaffolding or throwaway layers, interim representations upon which desired persistent architectural layers are constructed. Other layers may be used as "envelopes" to facilitate incremental phasing of new assets and subsystems into existing systems.

- Consider test plans as assets. White-box test plans are dependent on implementation technology choices and phasing incorporated into the ASSET IMPLEMENTATION PLAN. Test plans for phased releases specified in the ASSET IMPLEMENTATION PLAN that represent variants to be maintained under permanent configuration management might usefully be engineered as "test assets" in their own right. Test plans, test cases and test results can be stored explicitly in the asset base, where they can provide auxiliary examples and help to document the intended behavior of the assets. These test assets can be integrated into asset qualification functions eventually used by asset base managers or asset utilizers in retrieving and evaluating existing assets and adding new assets to the asset base.

### Exit Criteria

- Implementation technology selections have been made for all assets to be developed as part of this *Engineer Asset Base* phase.
- A schedule for implementing assets has been developed that is doable given PROJECT CONSTRAINTS and the estimated feasibility for the assets involved.

### Validation and Verification

- Validate individual asset implementation plans for consistency with the features of individual asset specifications.
- Validate technology selection choices for over-all coordination and consistency. For example, standardized choices should be made for similar implementation approaches within the asset base. Not all assets need to be built with a generator; but three different generative tools should not be specified without good rationale for the diversity.
- Prototype to validate technology selections. A good rule of thumb would be to prototype at least one asset using each candidate technology considered.
- Use asset base during development. Asset developers should become utilizers of the asset base wherever possible (except where this violates constraints on component sharing imposed for architectural reasons, as noted above).

The final validation of the ASSET BASE ARCHITECTURE and ASSETS as a whole comes through utilizing them in an application. This can be considered a kind of acceptance testing.

## 7.3.2 Implement Assets

### Purpose

The earlier steps of the ODM life cycle are intended to result in specifications that constrain the potential variability in implementing reusable assets down to a tractable design problem. The software development methodology used within the organization can be applied to the task of implementing the assets themselves.

The primary result of this task are the ASSETS themselves. The long journey through the domain engineering life cycle that leads to these products has enabled them to be engineered with considerable advantages over products of less disciplined approaches. The specification facilitates unstructured browsing and ad hoc queries by asset base customers, since it is mapped to the termi-

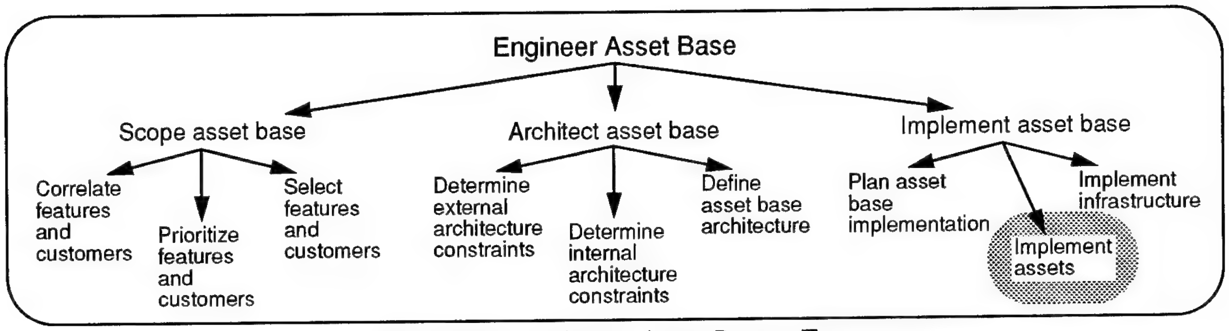

Engineer Asset Base
Scope asset base
Architect asset base
Implement asset base
Correlate features and customers
Prioritize features and customers
Select features and customers
Determine external architecture constraints
Determine internal architecture constraints
Define asset base architecture
Plan asset base implementation
Implement infrastructure
Implement assets

**Exhibit 69.** Implement Assets Process Tree

nology of practitioners in the domain. It is also amenable to automated support mechanisms for retrieval, configuration, and interchange.

With relation to the completed ASSET, the ASSET SPECIFICATION takes on more benefits. It can enforces the specific degree of information hiding judged appropriate by the asset implementor, so that the implementation of the asset can evolve over the lifetime of the ASSET BASE. At the same time, it can provide much more explicit detail about run-time performance, optimization criteria, etc. than a typical component specification.The purpose of *Implement Assets* as a core process within the ODM life cycle is therefore not to prescribe a particular software engineering methodology for asset implementation. Rather, the focus of this process is to address certain activities that must be integrated into asset implementation in order to utilize previous results of domain modeling and asset base architecture, and to address certain issues that will be faced regardless of what implementation techniques are used.

## Entrance Criteria

- Implementation technologies have been selected for asset specifications, and the necessary infrastructure (technical, organizational, and educational) for the selected technologies is in place. Heterogenous technologies can be applied.
- Architectural constraints on the assets to be created have been specified (e.g., loose vs. strong coupling to other assets).

## Inputs

- ASSET SPECIFICATIONS. These are the primary inputs to asset implementors. The specifications allow implementors to work reasonably independently, since architectural considerations have been factored into the development of the specifications.
- EXEMPLAR SYSTEM ARTIFACTS. Candidate artifacts for reengineering into assets.

## Controls

- ASSET IMPLEMENTATION PLAN. Details of the technology to be used and the implementation schedule for each asset.
- TEST AND VALIDATION PLAN.

## Activities

➤ Develop interface specification

Working from the individual ASSET SPECIFICATION, determine the degree of information hiding appropriate for *asset utilizers* and transform the specification accordingly. The specification should include any instructions for contexts of application, integration and adaptation, testing strategies, etc. Record the results in the ASSET INTERFACE SPECIFICATION. A separate interface level may be required for developers of other assets who will make internal use of the asset within the asset base.

Depending on the implementation technology selected, each feature-oriented ASSET SPECIFICATION may be elaborated into a set of conventional component interface specifications. One ASSET SPECIFICATION may be implemented by a single highly parameterized component, a set of component variants, or a generative system. If technologies applied evolve over time, these specifications will change but the original ASSET SPECIFICATION should remain fairly stable.

Just as some ASSET SPECIFICATIONS will not be developed further in various implementation stages, it is also possible to refine an ASSET SPECIFICATION into its ASSET INTERFACE SPECIFICATION without completing the implementation.

➤ Implement each asset

Apply the technology specified in the ASSET IMPLEMENTATION PLAN to implement the asset. This activity is essentially the interface to a large repertoire of techniques that we have classified into the broad supporting method areas of Component Development, Generator Development, Reengineering, and System Modeling described in Section 8.0.

A primary success criterion for the quality of the domain engineering process that leads to these activities will be the extent to which the models and other workproducts created serve the asset implementor in making effective use of this repertoire of techniques.

➤ Identify prototype artifacts

For each asset to be produced, systematically consider the resources to aid in the task that have resulted from previous tasks in the domain engineering life cycle. Consider the following types of resources:

- Artifacts examined and profiled during the *Model Domain* phase. Since existing artifacts will only loosely match the required feature profile, there may be several candidates. Where this would be a disadvantage for an asset utilizer, the asset implementor can use these multiple prototypes as contrasting templates or examples to help make alternative design choices apparent. (See the Guidelines below for more detail.)
- Other assets within the evolving asset base. The ASSET BASE ARCHITECTURE specifies which assets *must* be tightly or loosely coupled (to support various asset ensembles and configurations) and which must *not* be so coupled (to support separate selectability). Between these two constraints there are possibilities for leveraging assets, interim versions, supporting tools, etc. from the asset base development effort as a whole.
- Assets drawn from existing asset bases. This looks forward to an assumed scenario when multiple domain engineering efforts may have been performed or may be performed concurrently, perhaps in closely bordering technical areas. This scenario introduces constraints on

the process that have been reflected in the care taken in the *Define Domain* sub-phase of *Model Domain.* Here, the scenario presents opportunities as well as constraints.

➤ Develop asset supporting material

- Refine and encapsulate elements of the overall TEST AND VALIDATION PLAN to a specific ASSET TEST AND VALIDATION PLAN.
- Document all system artifacts consulted in engineering the asset. Depending on the closeness of match in the feature profile, these prototypes are now candidates for back-filling the engineered asset into the legacy systems concerned. This information will be of use for the ongoing marketing activities in Asset Base Management (beyond the scope of domain engineering.)

### Workproducts

■ ASSETS

Each ASSET has the following structure:

- ASSET INTERFACE SPECIFICATION. Expressed in terms of domain-specific features as documented in the DOMAIN MODEL. The interface specification explicitly documents the intended ***scope of applicability*** for the ASSET, the range of application contexts for which it has been designed.
- ASSET TEST AND VALIDATION PLAN. Developed from the overall TEST AND VALIDATION PLAN, the ASSET TEST AND VALIDATION PLAN provides supplemental documentation of the contextual assumptions and semantics of operation for a single ASSET.
- ASSET IMPLEMENTATION. Can take the form of a software component, a parameterized template, a generative tool that produces an asset instance at the point of demand, or any of a variety of other forms.
- ASSET DELIVERY MECHANISM. Provides any needed support for delivering the ASSET instantiation to the customer at the point of demand. This can include documentation of suggested integration, validation, and tailoring strategies.
- ASSET SUPPORTING MATERIAL. Other supporting material for an ASSET, possibly including example applications, and a list of potential customer sites derived from the ASSET BASE DOSSIER.

### Guidelines

- Consider the spectrum of reengineering options. Artifacts identified as prototypes will typically require reengineering to conform to the asset specifications. Since assets may be drawn from any phase of the life cycle, asset engineering can at time resemble forward engineering, at times reverse and/or reengineering.

    *Example.* If both reusable designs and reusable code assets are to be developed, then reusable code components could be developed by *reengineering* legacy code artifacts, by *forward engineering* reusable design artifacts or assets into code assets, or some combination of these strategies. Alternately, design assets could be derived by *reengineering* legacy designs, from *reverse engineering* code artifacts or assets into designs, or some combination of these strategies.

Traceability information needs to have been maintained to locate artifacts with feature profiles close enough to the asset feature profile. Whether used as prototypes or not, each identified match with legacy artifacts provides information useful in later marketing of assets.

> *Caveat.* In accessing artifacts in this way, the asset implementor is essentially engaging in a variant of ***ad hoc reuse***, since the artifacts being reused have not been reengineered for reusability. The rationale, of course, is that asset implementor in the domain engineering context do this *once*, thereby producing the very reengineered assets required for more systematic reuse on an ongoing basis. But the implementor should bear in mind that at this particular juncture, the domain engineering process is exposed afresh to *all the risk factors in ad hoc reuse*.

On the positive side, any support technology developed to support ad hoc reuse (textual search, data mining, etc.) can all be employed as support here as well.

- <u>Make bootstrapping use of assets</u> built in earlier implementation stages. This will help achieve the greatest synergy and economy of joint development possible for the assets. It also serves as a form of validation for the asset specifications and the asset base infrastructure as it evolves.

- <u>Stick to the asset specification; don't "overbuild"</u>. The most important guideline in this part of the process is, in some ways, counter-intuitive: In implementing the asset, it is critical to stick precisely to the specification developed from the prescriptive feature model.

  In non-reuse based system development, there is rarely any justification for expending additional resources to "over-build": i.e., to implement a component in a way that exceeds specifications or requirements for the projected system-in-use context. Ad hoc approaches to reuse may circumvent this policy in a number of ways. Implementors may expend "hidden" labor to add additional functionality (i.e., "make the component more general/reusable/flexible"). The temptation will be stronger if it appears that such additional functionality will not disturb the performance of the overall system. Only in special applications like mission-critical software that must be formally verified will problems like "dead code" (within a given system context) or unutilized functionality be of concern.

  *Such overbuilding could be disastrous in the context of an ODM domain engineering process.* The whole point of the domain modeling and architecture effort has been to design the "envelope of variability" for an individual asset within the context of a well-structured asset base. Expanding the functionality of one component will change this overall design; this is the *wrong place* to make that decision.

- <u>Don't optimize merely out of habit</u>. In a related issue, many developers grow accustomed to making design decisions based on assumed priorities about which engineering qualities to optimize, e.g., performance. In a reuse context, there may be an ensemble of components specified to optimize for different priorities; e.g., space vs. time efficiency. This may be an unusual way to design software, but will make sense in the context of the asset base.

- <u>Validate general reuse guidelines against domain-specific constraints</u>. Many guidelines recommend building reusable components to be extremely robust in the face of invalid data. However, in an asset base specification it may be desirable to have a "lean" version of a component that makes strong assumptions about its system-in-use context, and thereby achieves a strategically important level of performance. This may appear to fly in the face of general design-for-reuse guidelines, but may be appropriate in a domain-specific context. In the ODM context, these general guidelines must be reviewed carefully for built-in assumptions that may conflict with the asset base design.

### Exit Criteria

- A set of ASSETS have been completed that provide adequate coverage for a specified stage of the ASSET IMPLEMENTATION PLAN.
- Each ASSET has been implemented utilizing the selected technology and in accordance with the ASSET SPECIFICATION.

### Validation and Verification

Formally validate that:

- The assets respect the internal and external architectural constraints specified in the ASSET BASE ARCHITECTURE.
- Each asset implementation meets its specification. A particular point of concern here is that assets have not been overbuilt, and there are no unanticipated overlaps or conflicts among assets.

For further validation:

- Integrate the asset into at least two applications within its intended scope of applicability. It is necessary to test not only the functionality of the asset but also that it fulfills its mission in terms of support for variability.

## 7.3.3 Implement (Asset Base) Infrastructure

### Purpose

The primary focus of this process to is implement mechanisms and procedures that provide access to coordinated ensembles of assets. The main thread of activity is transforming the ASSET BASE ARCHITECTURE into an implemented domain-specific infrastructure. In addition, the domain model (i.e., the language formed of domain-specific features) leads to domain-specific extensions to the general asset base infrastructure: beyond the scope of any one asset, but particular to the domain model upon which the asset base is founded.

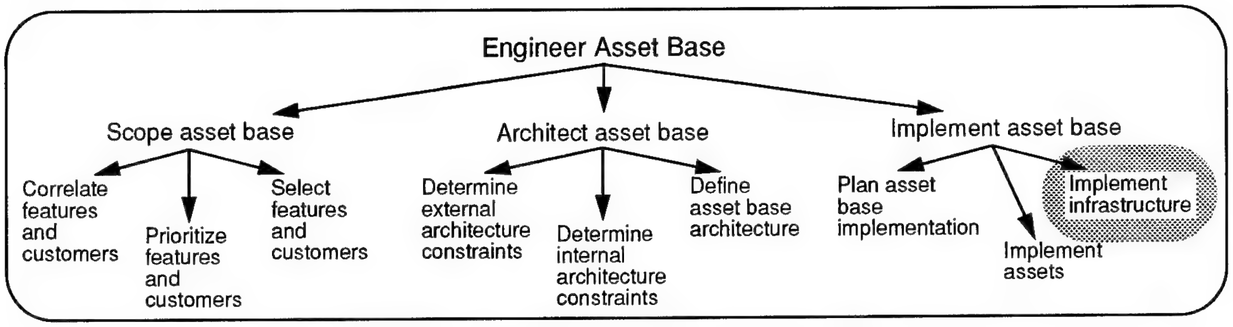

Engineer Asset Base
Scope asset base
Architect asset base
Implement asset base
Correlate features and customers
Prioritize features and customers
Select features and customers
Determine external architecture constraints
Determine internal architecture constraints
Define asset base architecture
Plan asset base implementation
Implement assets
Implement infrastructure

**Exhibit 70.** Implement Infrastructure Process Tree

The asset base infrastructure addresses the functionality required to assist system developers (asset utilizers) in:

- Being aware of the availability of an asset base covering a domain of interest;

- Searching the asset base for the asset(s) that match their system needs;
- Retrieving, adapting and integrating the asset into their development process and/or products;
- Contributing any appropriate feedback to the asset base with respect to their experience with the assets.

This process specifically addresses those elements of the asset base not confined to particular assets, and requiring domain-specific extensions to any general-purpose technology utilized for the asset base. It also addresses implementation tasks most appropriately performed by or in coordination with asset developers rather than as part of ongoing maintenance of the asset base.Other aspects of asset base infrastructure will be ongoing maintenance and sustaining activities that will be the responsibility of an asset base management group organization. (In CFRP terms, these activities would fall under Asset Management rather than Asset Creation.)

## Entrance Criteria

For retrieval infrastructure:

- General-purpose asset base infrastructure technology has been selected (a tailoring and infrastructure planning process).
- The DOMAIN MODEL has been completed. (See the notes under this workproduct in the Inputs subsection below.)
- The ASSET BASE ARCHITECTURE need not be complete.

For architectural composition:

- The ASSET BASE ARCHITECTURE is complete. Although only a subset of assets may get implemented within the project scope, it is important that the composition mechanisms in the ASSET BASE INFRASTRUCTURE be robust enough to grow towards complete implementation of the elements of the architecture.

## Inputs

- DOMAIN MODEL. Domain features form the basis for the query/retrieval interface to the asset base.

  NOTE: We don't want this interface constrained to include only PRESCRIPTIVE features, because we want asset base customers to be able to request feature profiles that are not (yet) included in the asset base.

  Some of these may be allowed for in the architecture but not yet implemented; some may be constructible from elements in the asset base but are not included as direct support. Over time, if requests for this particular feature profile are frequent, the asset base may be evolved to include them. The requests may also "stretch" the architecture in some ways.

  The requests may also be satisfied by a different asset base drawn from the same feature model. Coordination would be desirable; but a common feature language for interface should aid in this effort.

- ASSET BASE ARCHITECTURE. Used as the basis for implementing mechanisms that support interactions and interdependencies among individual assets.

- ASSET BASE CUSTOMER MODEL. A component of the ASSET BASE MODEL that describes the intended customer contexts for the ASSET BASE. This is input to the processes for communicating the existence of the ASSET BASE to potential customers; and to the design of the retrieval infrastructure.

## Controls

- INFRASTRUCTURE IMPLEMENTATION PLAN. The INFRASTRUCTURE IMPLEMENTATION PLAN includes technology selections and scheduling constraints for infrastructure development relative to development of ASSET releases. Used to guide the detailed performance of the task.
- TEST AND VALIDATION PLAN. Since the infrastructure will support separate configurations of assets, it may need to support incremental releases that will be used during testing and validation but not preserved as separately selectable configurations by end utilizers of the asset base.

## Activities

Many aspects of asset base infrastructure development will rely on general-purpose (i.e., non domain-specific) representation systems and tools. Selection, acquisition and/or development of such technology are considered within the domain of supporting methods in the area of Asset Base Infrastructure Technology. There are also dependencies with other supporting methods. described in the paragraphs below. See the appropriate sections in Section 8.0. The following activity descriptions focus on the main *process* aspects of this task.

➤ Extend Base Infrastructure Technology

General asset base infrastructure technology will usually need tailoring and/or adaptation to requirements for the specific domain. Working from any infrastructure selection decisions incorporated in the ASSET BASE INFRASTRUCTURE IMPLEMENTATION PLAN (some may be made in this task), and infrastructure requirements stated or implied in the ASSET BASE ARCHITECTURE, ASSET BASE CUSTOMER MODEL, and the original DOMAIN MODEL, make the domain-specific extensions required.

Making these extensions may involve developing some integration with the Taxonomic and/or Conceptual modeling supporting methods employed during the *Model Domain* phase, or the Software Architecture Representations selected for structuring the asset base in the *Architect Asset Base* sub-phase.

➤ Implement Architecture Mechanisms

Working from the ASSET BASE ARCHITECTURE, implement support mechanisms that will allow separate access any of the specified ASSET ENSEMBLES. These may include any or all of the following aspects:

— Specification/selection of multiple sets of assets;

— Delivery of configurations, with the appropriate asset-to-asset interfaces, glue code, etc. (The latter may need to be generated on demand, e.,g., if combinatorial configurations are possible.)

— Development mechanisms to verify that dependencies have not been engineered into assets that violate the architectural constraints imposed.

- — Configuration management support for ongoing tracking and management of various configurations as required.
- — <u>Point-of-utilization configuration of ASSET ENSEMBLES</u> specified as separable in the ASSET BASE ARCHITECTURE.

  *Example.* Simple examples of such mechanisms are the Install procedures provided with most large commercial software packages. These have evolved into separate support applications in their own right that seamlessly handle many of the chores of installation and the complexities of variable platforms and configurations for customers. Each usually provides a set of options such as “minimum”, “standard”, “full”, and “custom” installations; for the latter, users are walked through the choices of what modules to load. Note that there are usually constraints on what modules require the presence of other modules to execute; and there is also usually a sizable core module that cannot be customized by the user. Thus such support tools reflect underlying architecture choices made for the product.

Consider use of Generative Technology supporting methods for the infrastructure as well as for individual assets; e.g., for smart configuration management, making inclusion of required components relatively transparent to asset utilizers.

➤ <u>Implement retrieval and qualification support mechanisms</u>

- Support for *retrieval* of assets based on requests from utilizers expressed in terms of the domain features of the DOMAIN MODEL. This could include support for queries, browsing, navigation, and guidance in selecting assets.
- Support for *qualification* of assets with respect to their specifications and placement in the asset base.
- Consider additional requirements, such as support for *interchange* of assets with other asset bases, as the technology matures.

➤ <u>Support technology transfer planning</u>

Using the Asset Base Customer Model, specify any appropriate aspects of strategies for communicating the availability of specific assets and the asset base as a whole to the relevant applications groups. Address aspects requiring access to detailed knowledge about the implementation strategies of the assets.

## Workproducts

■ <u>ASSET BASE INFRASTRUCTURE</u>

Any portion of the environments, tools, documentation, and procedures required for ongoing integrated management of the ASSET BASE that are most effectively developed in tandem with the specific ASSETS of the ASSET BASE. These may include:

- Domain-specific extensions to general infrastructure mechanisms and procedures selected.
- Technology-specific extensions to the ASSET BASE ARCHITECTURE, reflecting additional constraints and semantics imposed by technology choices on the architecture.
- Inputs to the Technology Transfer Plan. The Technology Transfer Plan is developed outside

the domain engineering life cycle, in Asset Base Management. Asset Base Management receives and transitions the ASSET BASE to ASSET BASE users.

## Guidelines

- Align with organization's software engineering practices. As with assets themselves, implement the asset base infrastructure using methods that are consistent with the organization's software engineering methods.
- Don't co-opt asset management tasks. Don't take on tasks more appropriately performed by those people who will manage the asset base on an ongoing basis. The ASSET BASE CUSTOMER MODEL provides a reasonable basis for developing a technology transfer plan for promoting the asset base. However, since these activities will be ongoing they are more appropriately performed, hence planned, by Asset Base Management.
- Let asset implementors use the retrieval infrastructure to find previously built assets from earlier phases and to check interactions with assets being built by other developers. This serves as early validation of the infrastructure.
- Be incremental about optimization. There is a big difference between providing *all* and providing *only* the functionality required for a given asset ensemble. Consider staging the infrastructure development if necessary. This staging strategy will be distinct from the staging of the development of assets themselves.

## Exit Criteria

- The infrastructure has been integrated with an initial set of assets.
- The retrieval infrastructure and other asset utilization support mechanisms are implemented and have been used by the asset implementation team.

## Validation and Verification

- Formally verify that each ASSET ENSEMBLE specified in the ASSET BASE ARCHITECTURE can be separately obtained from the ASSET BASE.
- Sample customers from the intended asset base market can successfully retrieve, adapt and integrate appropriate assets as necessary using the infrastructure.

More extensive validation could include the following:

- An asset utilizer specifies a feature profile for an asset within the scope of the domain, but not implemented in the (current) asset base.
- Asset utilizers requesting a feature profile outside the domain scope are effectively informed that the request is out of scope and do not conduct a wasted search.
- A different *Engineer Asset Base* initiative, based on the same DOMAIN MODEL, is able to reuse much of the retrieval infrastructure developed for this ASSET BASE.

# Part III: Tailoring and Applying ODM

Part II presents a detailed description of the core ODM process model. The model offers considerable guidance for performing the core ODM life cycle activities:

- selecting and defining a domain of focus
- modeling the range of potential variability within the scope of the selected domain
- engineering an asset base that satisfies some subset of the domain variability, based on the needs of specific target customers

Although ODM encompasses all of domain engineering in this way, the core method offers prescriptive and detailed guidance only within a relatively narrow scope, focusing on activities that are unique to *domain* engineering and for which ODM offers a unique or distinctive approach. Other activities that are under the umbrella of the ODM life cycle because they are needed for domain engineering, but not unique to it, or for which there are a number of valid methodological options from which to choose, are relegated to the category of ***supporting methods***.

The scope of the core ODM method is further constrained to primarily address activities that are *necessary* for each of the major life cycle activities above. A number of value-added capabilities could be incorporated into ODM to address particular needs, but some organizations that do not have those needs could find the extra capabilities burdensome. Thus, ODM relegates such capabilities into a set of process ***layers*** that can optionally be selected by organizations that require them. Exhibit 71 shows how ODM has been designed in terms of these layers and supporting methods.

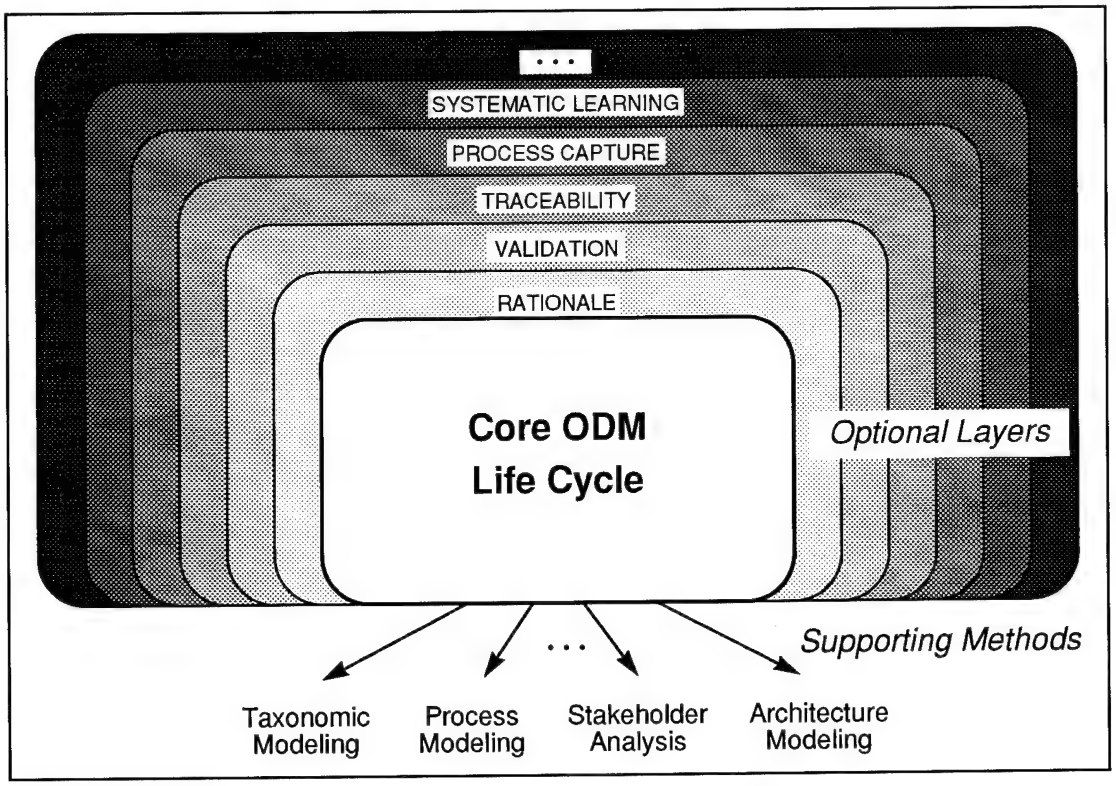


**Exhibit 71.** ODM Layers and Supporting Methods

The advantage of designing ODM in this way is that it enables clear definition of the boundaries and interface points between ODM and other methods and techniques to make ODM as tailorable and adaptable to differing project and organization contexts as possible. The burden that accompanies this advantage is that the tailoring activity itself can be a substantial challenge. Some of the specific objectives (and, viewed in another way, challenges) of the tailoring process include:

- Defining the role of ODM within a specific overall software engineering process or life cycle.
- Configuring the specific domain engineering processes to be performed and their interactions.
- Selecting and integrating the specific set of supporting methods that will be most effective and applicable within the context of the projects, organizations, and domains in which ODM is applied.
- Similarly, selecting and integrating the added layers of capability that are needed across ODM to meet project or organization objectives.
- Selecting and integrating the tools that are needed to support the selected methods and layers, as well as the core ODM techniques.

The purpose of Part III is to offer some general guidance in how to tailor and apply ODM for use in specific domain engineering projects.

- Section 8 provides brief descriptions of a number of categories of supporting methods and can thus assist in identifying and selecting specific methods.
- Section 9 plays a similar role for the optional ODM layers.
- Section 10 offers significant practical guidance for tailoring and applying ODM, ranging from guidelines for staffing and budgeting domain engineering projects to suggested strategies (with examples) for how to address some of the tailoring issues noted above. This section offers a number of practical hints and suggestions reflecting the experiences and lessons learned from past and current applications of ODM.

Unfortunately, in this version of the guidebook, the information provided in these sections is incomplete relative to what was planned. Treatment of supporting methods and layers is uneven; significant useful information is included for some methods and layers, while little or nothing is said about others. The guidelines offered in Section 10 are similarly uneven. These imbalances reflect only on the amount of time that was available to write the guidebook, rather than on the relative importance of particular methods, layers, or areas of guidance. The next version of the guidebook will rectify these problems.

Note, however, that detailed and extensive tailoring guidance is beyond the scope of the guidebook. In particular, the issue of automated tool support for ODM will not be addressed here. In this and other areas, supplementary guidance will appear in separate documents in the coming months.

# 8.0 Supporting Methods

In this section, we briefly discuss aspects of some supporting methods in relation to their role in ODM-based domain engineering.

What is a Supporting Method? In the ODM context, we encapsulate as a supporting method any body of techniques, methods, and tools critical to the accomplishment of overall domain engineering project goals, but

— not unique to domain engineering; and/or

— highly dependent on the specific needs of each organization applying ODM.

Applicable methods used in other kinds of engineering tasks, or other disciplines entirely, are usually well-documented elsewhere. ODM's interactions can be considered as "subroutine calls" to these well-defined external disciplines. Other supporting methods involve sets of specialized techniques highly relevant and specific to domain engineering but varying according to the organization, the domain, or the representations and tools selected. Documentation of such methods in sufficient detail is beyond the scope of this document. A supporting method may include any or all of the following elements:

— a modeling representation language and/or notation;

— a specific set of methods, skills and competencies;

— a repertoire of best-practice or recommended options for the area of concern; and

— specific tools, technologies, training materials, specialists, and a technical literature.

Supporting Method Areas. Though there are varying reasons for separating supporting methods from the core ODM life cycle, the interface to supporting methods in ODM has been structured in a consistent way. This section clusters supporting methods into a set of broad categories or areas that represent an initial attempt to identify the supporting methods most essential for ensuring the overall success and quality of an ODM project. The categories are provisional only; there may be significant overlaps between them, and there may be useful supporting methods needed for ODM that are not included here. The ODM process model and method are not dependent on this particular categorization.

Selecting appropriate supporting methods will be one of the primary infrastructure planning tasks of the domain engineering project planner. There are usually many approaches possible for each area, and the factors to be considered in selection of the methods will usually extend well beyond the scope of the domain engineering project. The intent in this section is to provide brief descriptions of the range of possible options in each supporting method area and to identify particular constraints or considerations relevant to the role of the method within ODM. The purpose is not to make general recommendations or comparisons.

Descriptions. The descriptions in the following subsections will discuss some or all of these aspects of each supporting method area, as appropriate:

- *Description*: The general problems and tasks addressed by the supporting method. A brief survey/typology of the range of options, and/or a set of illustrative examples may be provided. These are not exhaustive catalogues nor prescriptive sets of recommended choices.
- *Role in ODM*: The general relationship between the focus of the supporting method and the overall goals of domain engineering; and the specific tasks within the ODM life cycle where

interaction with this supporting method is of most importance. How critical is the supporting method to ODM? There are four basic reasons for considering a given supporting method.

— Must Do: If this supporting area remains unaddressed your project is at risk. For example, taxonomic modeling techniques are intrinsic to domain engineering. If you do not explicitly select a method you will still be performing these activities, but probably in an ad hoc way. These are supporting methods because the specific taxonomic representations used, the degree of formality involved, etc. will be project-specific.

— Nice to do if you can: The method area may not be appropriate for all projects, depending on their scope and resources. Access to personnel with these skills would generally enhance a domain engineering team. Results of work in these areas might prove useful data for domain engineering; similarly, domain engineering results could be useful to work in the supporting method area.

  For example, generative techniques are an important part of the implementation repertoire for reuse, and the ODM method helps in discovering opportunities for applying these techniques. But it is quite possible to build a well-structured asset base that will be utilized effectively that does not employ generative techniques. The STARS Demo project was organized at a high level as a coordination between a domain engineering and a system reengineering team, with independent but mutually supporting objectives.

— Synergy possible: For example, suppose your project is examining legacy designs during Acquire Domain Data and you have only code for some representative systems. If there are other reasons for the organization to do reverse engineering of those systems (e.g., to bring legacy systems into line with company standards for design documentation) then these tasks may dovetail quite cost-effectively with domain engineering.

— Conflict Possible: If you're performing activities related to this supporting method you need to be in coordination; e.g., a possible overlap of a domain engineering effort with a process improvement campaign in the company.

- *Guidelines*. Constraints imposed by ODM on selection of specific methods, and guidelines on integration of the methods and techniques with the domain engineering processes described in this document.

Supporting Methods Discussed. The methods in the following subsections are arranged in the approximate order in which they are first encountered in the ODM life cycle.

[Note: As discussed in the introduction to Part III above, the treatment of specific supporting methods in this section is uneven; significant useful information is included for some methods, while little or nothing is said about others. This reflects only on the amount of time available to write this version of the guidebook, not on the relative importance of the methods.]

- ***Stakeholder analysis*** techniques are used throughout the *Plan Domain Engineering* phase, but particularly in the initial sub-phase, *Set Project Objectives.*

- The subsequent sub-phases, *Scope Domain* and *Define Domain*, represent aspects of the life cycle that are fairly specific to domain engineering; no specific supporting methods are included for these sub-phases. However, because of the strategic importance of domain scoping, and the methodological importance of domain definition, success in these activities will depend to a large extent on integrating the organizational and engineering disciplines represented by the supporting methods as a whole.

- ***System modeling***, ***reverse engineering***, and ***process modeling*** are all used in the *Acquire*

*Domain Information* sub-phase of the Domain Model phase. These are techniques familiar to most software engineers. In addition, a repertoire of other ***information acquisition*** techniques can be used in the *Acquire Domain Information* sub-phase that draw on a variety of non-software engineering disciplines.

- ***Taxonomic modeling*** and ***conceptual modeling*** techniques are used in the *Develop Descriptive Models* sub-phase.
- ***Innovation modeling*** techniques are used in the *Refine Domain Model* sub-phase, particularly in the *Extend Domain Model* task.
- ***Market analysis*** techniques are used in the initial *Scope Asset Base* sub-phase of the *Engineer Asset Base* phase of domain engineering. They play an analogous role here to the stakeholder analysis employed at the start of *Plan Domain Engineering* and the information acquisition techniques employed at the start of *Model Domain.* Stakeholder analysis focuses on the full range of stakeholder roles, information acquisition on informant roles, and market analysis on customer roles.
- ***Variability engineering*** techniques are used in the *Architect Asset Base* sub-phase. In addition, ***software architecture*** techniques are employed in this sub-phase in a dual way, both addressing the architectures of the systems into which assets will be integrated, the architectures of applications or sub-systems that will be derivable from the asset base, and the architecture of the asset base itself. Both of these techniques have a role in the *Plan Asset Base Implementation* task of the final sub-phase, *Implement Asset Base.*
- The last four supporting methods discussed here all concern various ways of implementing assets and asset base infrastructure. They include ***reengineering*** techniques for transforming artifacts into assets, ***component development*** techniques for implementing software components designed explicitly for reuse, and ***generator development*** techniques for applying transformational and generative technologies to asset implementation; all applied primarily in the *Implement Assets* task. Finally, the *Implement Infrastructure* task employs ***asset base infrastructure technology***, underlying mechanisms for supporting integrated management, retrieval and evolution of assets.

<u>Interdependencies</u>. Each of these supporting method areas may have interdependencies with core ODM processes and workproducts including:

— results needed from supporting method activities to perform core ODM tasks;

— ODM workproducts that can aid in the process of selecting an appropriate supporting method in a given area; and

— workproducts ODM contributes as inputs to supporting method activities.

In addition, there may be interdependencies *between* certain supporting methods that need to be considered in planning the domain engineering project as a whole. For example, while in theory the system development methodology used to implement software assets in the *Engineer Asset Base* phase can be selected independently from the methodology used to reverse engineer legacy artifacts into a common format for comparative modeling, there are probably pragmatic reasons to make sure a common methodology is used in these two activities. In some cases, examining the overlap between two supporting method areas helps to underscore particular challenges presented in domain engineering. The following paragraphs briefly discuss one such case, the potential overlap between system modeling and process modeling techniques.

Software developers create software applications that end-users execute to perform their work in operational environments. (ODM formalizes these notions as the system-in-development and system-in-use contexts, respectively.) Typically, system modeling techniques have been used to represent data and processes in the operational environment. Process modeling largely evolved out of system modeling techniques, but has more recently been applied to modeling the tasks of the software developers' environment itself. In principle the same representations could be used to model both kinds of environment, but in practice this is rarely done. Also, often different people are concerned with these separate modeling tasks for different reasons: e.g., system modeling to build applications; process modeling to do process improvement.

Many of the most interesting and potentially useful opportunities in domain engineering lie in the linkage between these two environments, however. Opportunities for reuse of both software products (e.g., components in the traditional sense) and software processes (e.g., tools that automate domain-specific software development tasks) become visible only when working with an integrated picture of both environments in interaction. Neither system modeling nor process modeling techniques are oriented towards representation of both these environments in an integrated way. One particular difficulty lies in representing the actual software applications produced, which appear as a kind of output in the developers' environment, but serve to encapsulate processes when executing in the end-users' environment.

Thus both system modeling and process modeling are listed as supporting methods to ODM, and the method can be adapted (with some constraints) to a wide variety of specific representations and methods in either of these areas. But the quality of domain engineering results will increase to the degree that the methods selected can be smoothly integrated (at best, if a single method is used in both areas). In addition, extensions to the state of practice in both method areas will be needed in order to obtain the full benefits of an ODM-based approach to domain engineering.

This is one extended example of the many inter-dependencies between the supporting methods area. Further relationships of this kind may be discussed briefly in some of the following sections. A more thorough explanation would need to be part of a full guide to ODM tailoring and project planning, which is beyond the scope of this document.

## 8.1 Stakeholder Analysis

### Description

Stakeholder analysis techniques are employed in situations where it is important to identify all the people who are potentially affected by a particular issue or decision. Typical kinds of situations where these techniques can be useful might include:

- change management in large organizations, where mandates from official lines of authority are not sufficient to guarantee that change will be adopted and sustained;
- product-oriented companies seeking to move beyond a traditional focus on vendor/customer relations to explore broader value-chain relations and alliances;
- issues requiring coordinated actions or decisions by multiple individuals or organizations where there is no central, hierarchical control or authority (e.g., negotiation, conflict resolution, political decisions, etc.)

Though there are many techniques and approaches, most stakeholder analysis techniques share the purpose of exploring more sophisticated ways of establishing relations with a main producing or initiating organization. A key task is revealing "hidden" stakeholders, who may provide valuable information from unique perspectives, and who may be affected by decisions in ways that would otherwise not be adequately considered. Thus many methods and practitioners strong emphasize techniques that “de-contextualize” information and assumptions implicitly held by participants, and work out of a strongly held ethical stance that advocates social and cultural equity over support of status quo power relations in organizations. The ODM method has been strongly influenced by these approaches.

Stakeholder analysis plays a part in many broader disciplines, such as Total Quality Management (e.g., quality function deployment approach), business process re-engineering and organization re-design. One emerging movement that puts central emphasis on stakeholder analysis is Future Search Conferences, best documented in [Weis92]. Future searches are intensive conferences (usually two to three days in duration), carefully planned and structured to bring the most diverse set of stakeholders possible together for rapid progress in finding common ground and generating ideas for future action. (It is interesting to note that the term “domain” is used in this work to denote complexes of interacting stakeholder interests that involve multiple organizations and social groups [Tris83].)

## Role in ODM

As discussed in the detailed descriptions in this document, domain engineering initiatives share many aspects with situations in which stakeholder analysis has been applied:

- Domain engineering, as part of a larger shift to reuse-based software development (or, more broadly, to a megaprogramming paradigm) involves significant dynamics of organizational change.
- Innovative approaches to reuse, like generative techniques or shifts of features across binding sites and contexts, can help to re-define and broaden the value chain of the software life cycle.
- Domain models and asset bases should eventually evolve into frameworks that support new business models, for new types of organizations. These will require techniques to identify a diverse set of stakeholders that span current organizational boundaries.

Stakeholder analysis techniques are used most intensively in the first task in the ODM life cycle, *Determine Candidate Project Stakeholders*. They will be useful again after the domain of focus is selected, in the *Orient Domain* task, in identifying the stakeholders in the domain as distinct from the project. They will also be applicable in the *Plan Data Acquisition* task, for identifying a rich set of informants, and in the *Correlate Features and Customers* task, for identifying asset base customers. In the latter-mentioned tasks, they will overlap to some extent with information acquisition and market analysis techniques respectively.

## Guidelines

- <u>How essential are these techniques to ODM</u>? Stakeholder analysis techniques mostly take the form of methods practiced by specialized consultants. Some input from consultants trained in these techniques early in the Plan Domain Engineering phase could have major impact on the strategic soundness of the project. If this is not possible, some reading about techniques will help you be aware of those tasks where stakeholder analysis is going on (whether formal or informal). Consider leveraging any resources in the company where these skills might be available.
- <u>Selecting methods and practitioners</u>. The most useful approaches to stakeholder analysis for

ODM projects will emphasize facilitating dialogue among stakeholders, rather than mechanical or analytical decision processes. The latter are appropriate for processes that are well-understood and mature; domain engineering is still a young discipline, and some flexibility is required. However, use concrete, visual tools and guidelines wherever possible for orderly representation of trade-offs among stakeholder interests -- conflict, synergy, etc. (For examples, see the templates shown in Exhibit C-1, Project Stakeholder Roles, and Exhibit C-2, Candidate Project Objectives.)

- <u>Categorize stakeholders into meaningful ways</u>. Consider the roles and relations that matter most to your organization. Draw on work in the reuse community on reuse-specific roles. The CFRP provides one useful starting point [CFRP93b, Crep95a].

*Organizational Consulting*

Success in stakeholder analysis relies on a more fundamental set of skills that form part of the general discipline of organizational consulting. The parallels between organizational consulting and domain engineering are instructive. Three central concerns in domain engineering are directly analogous to classic organizational consultancy challenges:

- <u>Presenting and underlying problems</u>. Though usually brought in to help solve problems (i.e., something is not working) consultants have learned that the initial problem around which they are called in, the so-called presenting problem, rarely proves to be the real or underlying problem that needs to be confronted for the health of the organization. Much of the consultants value lies in the ability to discern and diagnose that underlying problem, and to articulate it to the organization in a way that engenders effective action.

  In a similar way, a domain engineering project is often organized around a domain as defined by the stakeholders in the organization. If the domain could be adequately described as a system family or product line given the organizations current structure, then it would rarely be called a domain. Use of this term flags a perception on the part of people in the organization that some new organizing structure for software engineering needs to be found. If they already knew the best way to do this, there would be no domain engineering project. But, as in consulting, the presenting domain recognized by the organization may not turn out to be the best definition for the domain.

- <u>Influence without authority</u>. While having no direct-line management control, consultants must aid decision-makers in recognizing problems and must recommend solutions, even when this involves confronting managers about problems in organizational process, management style, or interactions. In order to get their recommendations accepted, the technical expertise of the consultant must be combined with a set of organizational consulting skills and competencies.

  In a similar way, domain engineering projects will often be initiated for precisely those opportunities within organizations that fall in between the boundaries of existing management structures: e.g., across established product lines, across separately-funded contract efforts, etc. To be successful, the asset base must satisfy multiple stakeholders with competing needs and values. However, reuse practice can rarely be merely mandated from above. Domain engineers must thus get their solutions, or their parts of solutions, adopted but may have no direct authority to make this happen [see Sche87b, Chapter 2].

- <u>Explicit contracting</u>. Largely because of the two issues mentioned above, a consultant's role becomes problematic unless the objectives, boundaries and expectations for the work and the consultant's role within the organization are established clearly and explicitly at the beginning. This is known as the *contracting* phase in the consulting task.

Clear contracting is just as essential for the domain engineering project, for many of the same reasons. The ODM process model has incorporated the importance of contracting in the first sub-phase, *Set Project Objectives*. This is the motivation for making objective-setting internal and not external to the process; and this is the real challenge in the activities of that initial sub-phase.

- Organizational Development. Organizations grow and adapt according to predictable stages. Many theorists assign different names, stages, and key challenges to this development. How an organization chooses to work as well as its strategic choices are influences by its circumstances, its competencies, and its stage of organization development.

  The notion of organizational development is of direct relevance to domain engineering, particularly in setting project and organizational objectives. Software-intensive businesses are typically experiencing classic growth and maturing dilemmas at the point when they commit to software development process improvement and anticipate high reuse of their technical efforts. Such dilemmas bring a host of forces regarding performance management, lateral communication, cost management, and job design that can support or hamper the process and adoption of even extremely robust domain engineering results.

In light of these parallels, any domain engineering project would do well to get some training in consultancy and technology transfer principles. An excellent starting resource for getting acquainted with these issues in a practical way is Peter Block's *Flawless Consulting: A Guide to Getting Your Expertise Used* [Bloc81]. Modelers and managers should also be familiar with typical and predictable stages of organization development and how they may influence a domain engineering project life cycle [Liev80, Grei72]. At a minimum, someone with these skills should be available as a consultant and coach to the team (i.e., to the "domain engineers as consultants").

## 8.2 Information Acquisition Techniques

### Description

A large repertoire of methods for information acquisition are applicable in an ODM approach to domain engineering. Artifact analysis in a software engineering context is closely allied to reverse engineering, and will not be discussed in further detail here. A complementary set of techniques for eliciting information from people, broadly referred to as *qualitative* methods, will be the focus of this discussion. Qualitative data gathering is central in countless technical fields and social scientific disciplines. Some related areas in software engineering where these techniques have been applied include:

- Knowledge acquisition techniques developed or adapted for expert systems or natural language-based systems development;
- Scenario-based requirements elicitation techniques;
- Work on capture of design rationale, e.g., protocol analysis of the programming or software design process.

In addition, a wide body of ***ethnographic*** techniques have been developed for observation, analysis, and interviewing. These techniques will be the focus of this section. They have been influential in the development of ODM, and are a particularly rich set of techniques for acquiring the information needed for doing domain engineering following the ODM approach. There have been highly successful projects that combined trained ethnographers and human factors researchers with software designers [Brow91, Lind93].

These techniques are important in domain engineering because developers embed numerous tacit assumptions within software artifacts, usually unintentionally; these implicit contextual assumptions only become visible (sometimes disastrously!) when reuse is attempted in another context. Thus a good domain modeler should be able to play the role of "techlorist," especially during descriptive modeling activities in the ODM life cycle. This requires a set of skills not usually considered part of the software engineer's discipline.

Some specific forms of ethnographic and related information-gathering techniques include:

- *Sociological or anthropological fieldwork* methods seek to develop a picture or description of the linkage between human behaviors and shared cultural beliefs and assumptions, by studying artifacts, rituals, and observable events within the community of study. Interactions of field workers with the cultural insiders help make explicit the tacit cultural knowledge that is so obvious that practitioners never really notice it any more. See Spradley on ethnographic interviewing and observation [Spra79a, Spra79b], and Schwartzman on ethnographic analysis [Schw93]

- *Linguistic analysis* attempts to capture specific lexicon and language usage within a given language community. (It is interesting to note that the term "domain analysis" also occurs in this work; here, "domain" refers to a general topic area within which specific lexicon terms are collected [Spra79a].)

- Eliciting *stories* is a technique increasingly being used in organizational settings to assess and understand organizational cultures [Boji91].

- *Clinical intake* or *evaluation interviews* attempt to tap individuals' motivations, beliefs and common behaviors from a psychological point of view. These methods of eliciting data can be separated from the diagnostic categories sometimes used to label pathologies, and have been compared to ethnographic fieldwork [Sche87a].

- *Appreciative inquiry* is based upon the assumption that you get what you pay attention to. e.g., if you focus on problems or pathology, you get problems or pathology; if you focus on what works well and is beneficial, you get positive stories about the work context. This approach is increasingly being used to assess organizational receptivity to change. It doubles as a change intervention in raising awareness about what is positive. The approach is especially useful in focus group interviews in which participants are defensive or unsure of why the modeler is observing and asking questions. [Shri90]

- *Journalistic interviewing* (as practiced a la Bill Moyers, etc.) is another approach to eliciting contextual data. This approach differs from others in that getting a story and creating interest for an audience is often more important than being true to the language and/or beliefs of the informant. This approach is not recommended, but until modelers have studied other methods in some detail, differences in technique, role, and underlying assumptions may not be apparent.

- *Phenomenological methods* attempt to create a degree of objectivity in observation and meaning-making by deliberate suspension of the field worker's agendas and intents in order to allow a phenomenon to reveal itself, rather than probing it. This is the opposite of journalistic interviewing: very non-invasive, but also not goal-directed. Though probably the hardest style to learn systematically, elements of this approach underlie all non-directive observational techniques, including the psychological and ethnographic approaches. (See, for example, [Zubo88], Appendix B, "Notes on Field-Research Methodology"; also [Wino87], Chapter 3, "Understanding and Being.")

## Role in ODM

The primary potential role of these techniques is in the *Elicit Informant Data* task within the *Acquire Domain Information* sub-phase. Naturally, information will be gathered from stakeholders throughout the entire process; but it is this task where it being more systematic about data-gathering becomes critical.

Linguistic analysis techniques may be applicable as well in the *Develop Lexicon* task. Scenario and narrative-based methods become important in the *Interpret Domain Model* task, when modelers attempt to reconstruct the rationale for commonality and variability across representative systems. Appreciative inquiry and related techniques may be useful in the *Prioritize Features and Customers* task, when customer interest in various feature combinations, including innovative features, is assessed.

## Guidelines

- How essential are these methods in ODM? This will depend greatly on both the culture of the organization(s) performing domain engineering and the overall project objectives. In many engineering organizations there will be strong resistance to data gathering techniques that appear to be too non-directive and unfocused, even when these attributes are deliberate aspects of the techniques. If project objectives include use of the domain model as a means of capturing veteran expertise in the domain, or if the domain engineering team is organizationally removed from the application groups that will be the eventual utilizers of assets, some aspects of these techniques will be key elements to success.

- Budget for training and outside consulting. Obviously most software engineers will not instantly become phenomenologists when they join a domain engineering team. (Nor would this be desirable!) In general it will be advisable to augment the team with consulting specialists to train engineers in some basic ground rules for qualitative information acquisition, and possibly to participate directly in information gathering

  The need for training in these techniques may be in inverse relation to a person's technical background. The more expertise a domain modeler has in the domain of focus, the more he or she will need to take a disciplined approach to data gathering. They may be able to obtain data readily and absorb it quickly, but not know how to document it explicitly for others. Non-domain expert modelers may have an additional advantage in that they will be less tempted to impose their own biases about the "right" designs and components for the domain.

- Avoid methodological purists. Willingness and ability to blend and mix techniques, and a balanced orientation towards purity of methods and practical results are essential to this work. Adapting field work training from a research orientation to an engineering context will take flexibility on the part of specialists as well as open-mindedness on the part of the project team.

- Use the skills available to the team. Acknowledge that these tasks require special skills and assess the team honestly in terms of these skills. However, remember that not all these skills are scholarly and research-oriented. Many are practiced intuitively by successful writers, journalists, and professionals in many fields. An excellent source of talent in this area can be found in the technical writing/documentation groups of many organizations.

- Respect confidentiality. If necessary, advise informants not to share information that needs to be treated as confidential, if you can't guarantee you'll be able to do this; or if you suspect the informant is using the role as a "mouthpiece" for other agendas. This is one aspect of many complex ethical considerations

## 8.3 Taxonomic Modeling

### Description

### Role in ODM

### Guidelines

## 8.4 Conceptual Modeling

### Description

### Role in ODM

### Guidelines

## 8.5 Process Modeling

### Description

### Role in ODM

### Guidelines

## 8.6 Reverse Engineering

### Description

### Role in ODM

### Guidelines

## 8.7 Innovation Modeling

### Description

### Role in ODM

### Guidelines

## 8.8 Market Analysis

### Description

Markets are arenas in which individuals and organizations develop economic relationships with each other based upon the developing and deploying of products and services. Market analysis is the segmentation of markets, using a matrix of factors and based upon assumptions and predictions about the size and change of the profit potential available within a given market segment. These market segments help business define their product/service sets and set goals.

Technology markets are complex in that the products are objects, training, books, services, and even abstractions like software which conventional product or service descriptions do not adequately describe. Analyzing these markets may require techniques beyond traditional kinds of market segmentation by price, volume, geography, product etc. Additional factors like market saturation, competition among product offerings and pricings, and substitute products are also central to analyzing markets. In technology, such concepts as leaders, early adopters, and laggards are important market analysis concepts for targeting like groupings of customer needs within differing time windows.

Approaches to market analysis include the following examples:

- Geoffrey Moore's "Crossing the Chasm" approach is particularly useful for technology-intensive offerings because it analyzes markets in terms of receptivity to innovation investment [Moor91a].
- *Scenario analysis* is a structured and creative way to ask the question "what if" and thereby be more ready for future challenges. This is particularly useful for predicting future markets and/or making necessary investments to create markets [Schw91].
- *Mass customization* is an approach to product planning and product redesign that specifically addresses some aspects of variability possible in software-intensive application areas [Pine93].
- *Value chain models* are concepts and pictures for showing relationships of different stakeholders within an increasing change of value between producer and consumer. They are pertinent to market analysis because market expansion often means moving further up or down the value chain that your current customer.
- The role of a domain-specific focus in the business model of an organization is also closely related to work on organizational core competencies [Prah89,Prah90].

### Role in ODM

Some researchers have equated domain analysis to a form of market analysis, specialized to software products targeted specifically to application developers rather than end-users of applications. The CFRP offers one specific model for reuse value chains that can be useful in market analysis for domain engineering projects [CFRP93b].

There are several challenges involved in applying conventional market analysis techniques to domain engineering. These include the following issues:

- Handling the greater possibilities for variability introduced by software;
- The complex nature of the software development value chain; and

- The possibility of encapsulating reusable processes as well as products.

In the ODM context, the specific market analysis aspects of domain engineering are most directly employed in the *Scope Asset Base* sub-phase, and also, in a less direct sense, in the *Scope Domain* sub-phase.

**Guidelines**

In light of the issues discussed above, we offer the following guidelines and caveats with regard to market analysis supporting methods:

- Recognize the market analysis task. Simple increased awareness of the need to consider explicit customers in asset base engineering may be the most significant pay-off, almost independently of the specific methods chosen. It is also important to keep several customers in mind, however, so that the real trade-offs involved in engineering for reuse remain visible.
- Do not rely exclusively on any one method. In particular, adapt the methods to the specific needs of reuse-based software engineering.
- Do not expect robustness from hard quantitative methods when applied to estimating usage of software components. Supplement these techniques where possible with qualitative data derived from other supporting methods. Stay in dialogue with the stakeholders to ensure that the project is aligned with their needs.
- Recognize when whole products need to be delivered. Throughout this guidebook there has been a strong emphasis on separation of concerns: clarifying what belongs in the province of domain engineering methods and what belongs to supporting methods, etc. In the domain engineering process itself, there is a similar emphasis on boundary-setting and scoping throughout. This separation of concerns is essential to make domain engineering a tractable engineering problem.

## 8.9 Asset Base Modeling

**Description**

**Role in ODM**

**Guidelines**

## 8.10 Software Architecture Representations

**Description**

**Role in ODM**

**Guidelines**

## 8.11 Component Development

**Description**

**Role in ODM**

**Guidelines**

## 8.12 Generator Development

**Description**

**Role in ODM**

**Guidelines**

## 8.13 Reengineering

**Description**

**Role in ODM**

**Guidelines**

## 8.14 System Modeling

**Description**

**Role in ODM**

**Guidelines**

## 8.15 Asset Base Infrastructure Mechanism

**Description**

**Role in ODM**

**Guidelines**

# 9.0 Optional Layers

## What is a Layer?

In this version of ODM, the notion of layers is used primarily as a way of making a separation between core and optional aspects of the method. There are two main aspects to layers that differentiate them from supporting methods:

- Layers appear to be tailoring choices that have *pervasive* impact throughout the life cycle. That is, they affect aspects of how every process is performed and each workproduct is produced. Supporting methods, in contrast, seem to play a key role in particular tasks of the life cycle.
- Layers clearly extend into *optional* areas for domain engineering projects. Most supporting methods involve activities that must be performed, in some way, in order to accomplish domain engineering objectives. There are a range of techniques to choose from, and ODM is reasonably tailorable to different techniques. Optional layers will not be characterized in terms of different methods and techniques available. It is also not essential that they be incorporated to any particular extent.

Use this section as a checklist of issues to consider in planning a domain engineering project. Within most layers various options will be presented for projects that may want to extend farther than the minimal recommendations in the core process model. Each project team will need to consider the trade-offs of resources and benefits relevant to their initiative.

The layers discussed are:

- Validation
- Traceability
- Process and Rationale Capture
- Team Modeling
- Learning and Evolution

The first layer discussed, Validation, is incorporated into the structure of the main sections of this document as the Validation and Verification sub-heading within each task description. The subsection here serves in part to summarize general strategies to validation provided as task-specific recommendations earlier in the document. In addition, it extends the notion of validation to optional approaches that, in effect, raise the standards by which the quality of domain models and asset bases would be evaluated.

The other layers discussed have been referenced only in passing in earlier sections of the document.

[Note: As discussed in the introduction to Part III above, the treatment of specific layers in this section is uneven; significant useful information is included for some layers, while less (or even nothing) is said about others. This reflects only on the amount of time available to write this version of the guidebook, not on the relative importance of the layers.]

### Background

The area of optional layers for ODM is still being defined. It is a response to lessons learned with earlier versions of the method, where it became clear that attempting to do domain engineering in a systematic way can impose a considerable burden on projects, particularly when support technology for domain engineering is just beginning to emerge. Gradually, it became clear to those working on development of ODM that there was a certain amount of hidden context in the method itself, i.e., preferences and agendas that were in close alignment with, but not absolutely essential to, the primary objectives in domain engineering.

One example of such a preference is the linkage of domain engineering to the concept of the "learning organization" popularized by the work of Peter Senge and Innovation Associates [Seng90, Seng94]. The potential for synergy between domain engineering methods and this approach to learning and change in organizations is compelling. However, there will be many organizations willing to try a small-scale domain engineering pilot project that are not prepared to tackle the transition to being a learning company in the process. Accordingly, we have attempted to excise portions of the method that seem relevant *if* performing domain engineering in companies oriented towards a learning organization approach. These are included within the optional Learning layer.

### Caveats

The core process model presented in the main sections of this document thus represents a significant re-architecting of ODM. An equivalent amount of re-structuring will be required in the area of the optional layers presented here for project planners to be able to systematically apply them in various "separately selectable" combinations. In this document the layers presented are provisional only. In particular:

- Each layer represents values or concerns reflected to some extent in the core ODM process model itself. Elements included in the core are those deemed critical for achieving the stated objectives of the ODM method of domain engineering.
- There will be significant overlap between layers. For example, traceability and validation activities are interconnected in a number of ways.
- The sequence of presentation does not imply that earlier layers must be selected before later layers, or make any other implications about ways in which these optional process areas can be combined.

## 9.1 Validation

### Description

Internal Validation

Individual models can be validated according to certain internal principles, including the following:

- *Completeness*: does the model cover all required variability as observed in the exemplar set?
- *Validity*: is every concept in the model justified by some precedent within the exemplar set?

- *Consistency*: is the model structurally complete? Are all model inter-relationships valid?
- *Expressiveness*: is the model capable of expressing variability that would be considered significant by a domain expert/informant?
- *Model balance*: is the model structurally balanced? Are there comparable and consistent levels of detail throughout the structure?
- *Minimality (Parsimony)*: Is there economical use of basic concepts in the model? Are the number of concepts constrained enough to admit a relatively intuitive grasp?
- *Intuitiveness*: do model concepts seem natural, or forced/contrived? How good is internalization of the model without recourse to documentation?

In addition, separately developed models or other workproducts are validated through integration. Integration activities can be interleaved with model development at various levels of granularity, ranging from activities performed by individual modelers to integration through project-wide review meetings. As technology for domain modeling matures, many integration tasks should become better supported by automated tools.

Process Validation

Process validation is more likely to impact the modeling process than the models per se. It is closely tied to the Process and Rationale Capture layer. Some useful questions to ask include the following:

- Was the process of creating the model a valid instantiation of the planned process?
- Was the actual process documented?
- Did all joint modelers achieve consensus on the model?

Stakeholder Validation

- *Formal validation of individual models*: Independent of the accuracy of the data in the models, validate them for consistency, well-formedness, style, etc. Automated tools can be useful as aids.
- Validation through integration: This is a large part of the role of the *Integrate Descriptive Models* task.
- Empirical Validation: Validate models with respect to the data used to create the models. Solicit feedback from informants on the veracity of the data. Check linkage from data to models.

Extended Forms of Validation

The validation activities listed above can be thought of as basic "good housekeeping" forms of validation. Extended forms of validation are possible. For example:

- Validate with exemplars that were not representatives. Use this validation to determine the robustness of the models, given unexamined data? The significant factor in this validation step is that all exemplars were presumably at least looked at in deriving the domain definition.

- Validate against new exemplars. These are SYSTEMS OF INTEREST that were not considered in the original EXEMPLAR SYSTEMS SELECTION.

These subsequent levels represent a transition from what has been termed ***variability engineering*** to true ***domain engineering***.

### Guidelines

- Do internal validation before getting external feedback. Refine and validate domain engineering workproducts using internal criteria and the resources of the team before drawing in external domain experts for further validation. However, it is also wise to seek external validation before products look too finished, so that stakeholders who participate in the validation process can see that they have a voice and some ownership in the work. Modelers may get too attached to their models after spending greater degrees of effort in polishing them; and experts may be more reluctant to critique models that have the look of finished products. Factors such as the degree to which the domain expert is part of the team, schedule and resource constraints also come into play. Prepare the material as thoroughly as possible, but leave the presentation somewhat informal and open-ended, so that real feedback can be obtained.

- Use ongoing reviewers. Use some key informants who are also potential customers for periodic review throughout the modeling life cycle. This continuity of involvement helps gain buy-in and ownership from the reviewers and addresses technology transition issues while providing the necessary validation with a minimum learning curve for the stakeholders providing the validation.

- Broaden the review audience. In addition to these ongoing validation roles, use successively broader review audiences as domain engineering workproducts become more stable. In particular, try models out on some members of a potential utilizer group not previously tapped as informants. Use combined expertise in group validation and review sessions (e.g., experts not accustomed to speak with each other.

## 9.2 Traceability

### Description

### Guidelines

## 9.3 Process/Rationale Capture

### Description

A certain level of skill in process capture is necessary to achieve the basic results of domain engineering. By studying processes in the system in development context, modelers are able to discover opportunities for encapsulating reusable processes as well as products. Even if not specifically designed for reuse, a software artifact can be reused more predictably with documentation of the rationale behind its design.

There is a secondary layer to this emphasis on process and rationale capture in ODM, which focuses on capturing process data and rationale for the domain engineering project itself. The primary motivation for this process and rationale documentation is to preserve information that will facilitate iteration or backtracking in the life cycle when it is required. Iteration is a natural part of the process; it can reflect breakdowns or errors, but can also be a response to changes in availability of resources or even new opportunities created by successful phases in domain engineering.

This form of process and rationale capture in ODM is evidenced in a number of the core workproducts and processes in ODM discussed in the main sections of this document, including the following:

- separate documentation of candidate lists and selection criteria before selection tasks
- separation of domain-specific from resource-based prioritization steps

Many of the workproducts described as "interim" in this document thus serve a dual purpose. The primary purpose is to break the modeling process down into relatively small and well-defined activities. In addition, they provide checkpoints that capture the decision process at particular well-defined phases; each of these phases can become a roll-back point if certain project objectives or constraints shift or an impasse is reached. Keeping the various workproducts consistent does impose some hefty overhead which should eventually be addressed by better automated support.

A second level of process capture, which is an optional extension to core ODM, is oriented toward the needs of future domain engineering projects within the same organization context. The intent is to facilitate reuse of various domain workproducts in subsequent projects. For example, if a new domain is selected in a business area where the ODM planning process was carried out thoroughly, a number of workproducts will have been created that will allow the selection process to proceed much more rapidly, including the Domains of Interest, the Domain Selection Criteria, the Candidate Project Stakeholder Model and portions of the Domain Interconnection Model. With each subsequent domain modeling effort in a given business area, this reusable base of domain engineering resources should be expanded and refined.

One significant kind of reusable resource that will help improve the productivity of subsequent projects are ***starter models***: abstractions from domain-specific models that provide a useful starting point for models in new domains. These might include any of the following:

- Reusable DOMAIN ONTOLOGY MODELS that encapsulate specific choices in the area of Conceptual Modeling supporting methods, and are represented in formalisms particular to a Taxonomic Modeling supporting method. For example, an RLF model could be provided that defines the root category names for a specific set of concept models to be developed. Other projects could start from this choice of concept models and tailor as appropriate.
- A set of optional domain-specific concept or feature models. For example, a model of the Ada type hierarchy could be developed that could be used in structuring asset bases of unit test plans for Ada components. This model would not need to be re-built for subsequent projects.
- A set of templates, intended for adaptation and tailoring; or guidelines, checklists, generative tools, validation and consistency-checking tools, etc.

Selection and tailoring processes occur throughout the process model, with a recurring set of subprocesses: reusing the available resources, defining criteria for candidates, generating a candidate set, establishing characterization criteria, characterizing, ranking and prioritizing the candidates, documenting the selection rationale.

A final level of process capture, also an optional extension to core ODM, would take this documentation one step further. In this layer, an attempt is made to document changes made to the various workproducts over time. The purpose of capturing this information is to rapidly improve the domain engineering process and methods through project experience and lessons learned.

A good example of a process capture technique of this sort is a "Domain Boundary Decision Log", which documents the history of iterative decisions that shift the domain boundary over the lifetime of the project. Study of this log could reveal certain problems in the process, such as boundaries that oscillate back and forth over time and fail to come to closure.

This kind of process and rationale capture could allow for examination of the methods to improve the process. This information can be of value to individual modelers who want to improve their skills, to an organization attempting to tailor a domain engineering method suitable for their specific needs, and to the developers of the methodology.

### Guidelines

## 9.4 Team Modeling

### Description

Many aspects of the ODM method require different sorts of team interaction skills than those customary for groups of software developers. The modeling process works by continual interaction between centralized and distributed modeling tasks. Several groups may build related models relatively independently, then meet to integrate the models and resolve the inconsistencies. Model development is partitioned into reasonably separate tasks, which can be performed iteratively, in parallel or even in round-robin fashion, with circulation of personnel between different modeling and data-gathering tasks. This separate modeling strategy is necessary, even when models produced are conceptually linked together and could in principle be thought of as one large model. This is considered an optional layer because a team modeling approach can be used throughout the ODM life cycle. At every point where it is used, it can offer the following benefits:

- optimization of staff resources;
- acceleration of project schedule through parallel tasking;
- independent cross-validation, by having separate teams work on closely inter-related tasks and models;
- mitigation of limitations in support technology (e.g., for dealing with large models);
- better control of the implicit "cognitive bias" by determining the sequence and diversity of data that particular modelers will examine.

However, use of a team modeling strategy will also pose some challenges for the domain engineering project. More stringent integration activities are required to ensure the consistency of separately developed models and the traceability of elements across various models. Unlike software modules, that can have an unambiguous interface and a clear separation of concerns, related models will tend to overlap and have numerous interconnections, not all of which can be anticipated in advance. This requires a high degree of constant, detailed technical interaction on the part of the modeling teams. It also requires greater attention to process management, and greater levels of collaborative and listening skills (the latter closely related to the descriptive modeling mindset alluded to in the section on Information Acquisition techniques in Section 8). Thus, the other aspect of team modeling that makes it an optional layer is the degree to which teams are willing to address these skills. A good resource for exploring these directions is provided in the learning organization field, specifically in the core discipline of team learning [Seng90, Seng94].

### Guidelines

- Build Diverse Teams. Diversity in team structure can enrich the quality of domain models by providing validation from multiple perspectives; however, it may also complicate the management task.
- Level of Domain Knowledge. Both experts and novices with respect to knowledge of the domain of focus can add value in domain modeling. Certain ODM processes are better suited for the "naive", others for the "expert" domain modeler. One might think a naive modeler must do all the work of the expert modeler, plus more to come up to speed in the domain. For certain tasks, however, modelers' prior domain knowledge of the domain can create potential perceptual biases. Teams designed to maximize diversity in this respect can produce highly robust results, by including personnel with a range of familiarity with the domain, as well as those experienced in related or analogous domains who bring still other perspectives to the domain modeling effort.
- Peripheral Experts. There may be significant culture clash between applications developers and domain modelers. It may be appropriate to identify a set of expert reviewers "of first resort" for the project, welcoming but not depending on full-scale participation. For example, such a "peripheral expert team member" could be assigned the task of creating and validating the Domain Lexicon which requires extensive domain knowledge but not familiarity with the domain modeling representation formalism.
- Circulation of Personnel. Circulation/rotation of personnel between domain and application engineering teams, particularly for projects of some duration, will strengthen connections between the teams and a sense of buy-in by the applications groups. However, the learning curve on domain engineering projects can be particularly steep; so rotation that is too rapid can be counter-productive.

## 9.5 Learning and Evolution

### Description

Learning is fundamental to reuse. An approach to domain engineering that does not incorporate direct support for ongoing learning about the domain runs the risk of achieving only a very limited and static kind of reuse, based on a single snapshot of knowledge about the domain. In fact, domain knowledge is constantly evolving for individual practitioners, the organization, and the technical area as a whole. Domain models and asset bases can play an integral role in accelerating that learning process, but only if infrastructure is in place to reflect the learning in the evolving asset base itself.

As each asset is used on successive applications, a base of experience grows around that asset. This might be reflected in requested changes or enhancements from asset utilizers, or in the development of ancillary material (example usage, tips on adaptation, test suites, etc.) developed and shared by utilizers. A well-designed asset base infrastructure will support the incremental addition of this supporting material over time.

Domain Evolution

Domain models will evolve over time, in addition to the assets created and managed with these models. Changes can take place with respect to several different timelines:

- over the course of the initial domain modeling process;
- over the lifetime of the assets and the asset base;
- with the evolving level of knowledge about the domain within the organization and the technical community.

Evolution of domain models and asset bases may take the following forms:

- incremental migration from informal model semantics (e.g., a checklist, outline, or keyword list) to more formal model semantics (a faceted classification scheme, or inheritance-based model);
- transformations in the classification scheme;
- gradual inclusion of more heuristic knowledge to guide modeling choices;
- changes in implementation, e.g., changing a component-based approach to a generative approach;
- generalization: promoting domain-specific models or components to the level of infrastructure, or more horizontal usage. For example, a model of error conditions developed for one real-time domain may turn out to be largely transferable to other domains with similar attributes;
- aggregate components: in an asset base that begins by implementing a base set of components, a set of "widgets" that configure multiple components can evolve over time;
- consolidation, diversification, model reorganization, and other transformations at the same implementation level.

Learning

There are several "learning loops" involved in the domain engineering and broader reuse-base development scenario:

- Individual practitioners' learning about:
  - the specific domain (i.e., the individual becomes more knowledgeable about the domain);
  - project-specific domain engineering infrastructure (the individual becomes more familiar with the infrastructure available);
  - general domain engineering methods and tools (the individual becomes a more skillful domain engineer, who could transfer this skill to other organizations and domains).
- Organization learning about:
  - the specific domain (i.e., industry knowledge about the domain advances);

- project-specific domain engineering infrastructure (i.e., the infrastructure evolves with use);
- general domain engineering methods and tools (the maturity level of the organization advances).

- Methodologists' learning contributing to:
  - Individual analysts' skills evolving towards transferable expertise in teaching and practicing domain engineering as a discipline; and
  - Advancement of methods and tools (i.e., domain engineering methods and tools evolve through learning feedback from practitioners).

## Guidelines

# 10.0 Guidelines for Applying ODM

## 10.1 Project Planning

### 10.1.1 Allocation of Resources

The symmetry and balance of the ODM process tree may give the mistaken impression that the level of effort expended on planning, modeling, and asset base engineering respectively should be about equal. It would be more appropriate to say that the granularity of the activities increases the farther into the life cycle we get. Thus, though the planning steps are elaborated to the same depth in the model, they should not take as long, nor should they consume as many staff resources as subsequent phases. The proportions suggested by the process tree are better indications of the specific content that ODM (in contrast to supporting methods) offers for each activity.

### 10.1.2 Scheduling and Effort Estimation

The *Plan Domain Engineering* and *Model Domain* phases together should take no more than approximately one third of the time allocated for domain engineering as a whole. Within this time, the initial planning phase should take roughly one third of the time allotted (i.e., one-ninth of the total project period of performance). We say "initial" here because a certain amount of iteration back to planning activities will recur throughout the project; this is in addition to ongoing project management activities. These proportions will not hold, of course, in situations where the project is only intended to perform steps through domain modeling.

Within the planning phase itself, expect each succeeding phase to take longer; i.e., *Define Domain* should be allocated the most time. *Select Domain* next, and *Set Project Objectives* should be the shortest process of the entire life cycle. First, but not least: even if the objectives and stakeholders are clarified in a few meetings at the start of the project, it is vital that these meetings unearth certain dynamics in the project context that will otherwise almost certainly create enormous problems farther downstream. The nature of these problems are distinct enough from those encountered on any software project that we believe the level of detail here is warranted.

The planning phase, while shorter in duration and leaner in allocated staff, requires the highest level of skill in negotiating the organizational aspects of domain engineering. As with the initial contracting in a consulting situation, mistakes in objective-setting and expectation management at the outset can stymie the entire effort. In addition, domain selection, one of the critical leverage points of the entire process, is embedded within the other planning activities. This creates a project management dilemma. On the one hand, planning activities need to happen fairly quickly and with limited staff. Yet the organizational complexities are greatest at this point, and there is a need for someone with seasoned modeling skills to be involved. This suggests a need for access to consulting or non-line management staff resources for the initial planning phase.

### 10.1.3 Infrastructure Planning

TBD

## 10.2 Tailoring

The ODM process model can be thought of as a framework that will be tailored for each domain engineering project. Because of the inherent variability in the domain engineering process in different organization settings, this tailoring is an anticipated part of applying ODM.

### Tailoring Strategies

Tailoring can take three major forms:

- A ***specialization*** of the ODM process model applies tailoring techniques to produce a process model that is itself a framework, but one oriented towards a narrower organization context than the general ODM method. Most typically, this will be a process model developed for the needs of a particular organization. Examples are the domain engineering workbooks/guidebooks produced by researchers within Hewlett-Packard and within the Army CECOM/SED organization [Coll91a, ADER95a, DEGB95a]. Specializations could also be produced for ODM methods tailored to particular kinds of domains, or assuming particular choices of supporting methods and layers.
- An ***instantiation*** of the ODM process model applies the tailoring techniques to produce a process plan for a single specific project. This will be most appropriate for projects that are attempting to do domain engineering using a process-driven approach. Process-driven software engineering is another facet of the STARS megaprogramming paradigm, synergistic with but independent of the focus on architecture-centric, domain-specific reuse that ODM directly supports.
- The ODM process model can also be used to document ***process history*** for a project. It can be applied both to projects formally conducted as domain engineering efforts, as well as to projects that performed domain engineering in an informal way, or even projects not explicitly recognized as domain engineering at the time of performance, but satisfying the bounding criteria for domain engineering established in Section 3.1. An early version of ODM was used in this review capacity within Hewlett-Packard, as part of the RADAR (Reusability Analysis/Domain Analysis Review) process. Process histories can be used for process improvement within the organization, comparison of different projects, or for evolution, validation and refinement of the ODM method itself.

By combining the three modes of tailoring into an integrated cycle, the combined benefits of a process-oriented approach and a systematic learning approach can be obtained. A specialized tailoring of ODM is used as the basis for deriving a domain engineering project plan. The project is performed with some resources devoted to maintaining a process history (i.e., what was done as opposed to what was planned), metrics, and lessons learned. The results are used to refine the specialized process model for the organization, to improve subsequent project planning. At the same time, the lessons learned provide feedback to refine the general ODM methodology.

### Tailoring Transformations

Whether done to produce a specialization, an instantiation or a process history, tailoring of the ODM process model can involve a number of specific transformations. The most important of these tailoring strategies have been formalized in the structure of the supporting methods, discussed in Section 8, and the optional layers discussed in Section 9. In addition to supporting methods and optional layers, the following types of transformations can be applied to the core process model presented in Part II of this Guidebook:

- Deletion of particular tasks or workproducts, considered inappropriate or irrelevant for the organization or types of domains of interest, or not feasible given project resources;
- Addition of other processes or workproducts to the project plan, based on opportunities for synergy with other engineering objectives for the organization;
- Sequencing of tasks into a specific order of performance, including the possibilities of:
  - — overlapping tasks or performing them in parallel with multiple teams;
  - — iterating between or cycling through series of tasks; or
  - — performing tasks retrospectively, working backward through the ODM life cycle from workproducts created following other methods or via an informal process.

    This process is analogous to reverse engineering, but applied to domain engineering rather than system engineering workproducts.
- Renaming of processes or workproducts to better suit the background and culture of organizations attempting domain engineering projects.
- Restructuring the process model by splitting particular tasks to greater levels of detail, consolidating task descriptions, etc. The activity descriptions in the document are intended to be especially flexible; some activities have a transitional nature and could be re-allocated. As long as the underlying dynamics of the process are respected, the model should be fairly robust and flexible.

A complete guide to this tailoring process is beyond the scope of this document. In the following paragraphs, we offer a few snapshots of specific tailoring steps that could be made based on particular organizational scenarios. These descriptions presume familiarity with the ODM process model described in Part II of this document, and are for the purposes of illustration only.

*Example: Transition from Domain Modeling to Asset Base Engineering*

Once the *Model Domain* phase has produced a DOMAIN MODEL, there are a number of alternative scenarios for validating the model and for completing *Domain Engineering*; these depend to a large extent on the PROJECT OBJECTIVES. The DOMAIN MODEL itself will be sufficient to address some of these objectives; others will require moving on to ENGINEER ASSET BASE. Each of the following scenarios can be treated as a possible strategy for validation and verification, or as a desired end use of the results of domain engineering.

1) *Use the domain model directly as an asset.* The domain model can be used as a basis for maintaining legacy systems in the domain, educating new domain practitioners, etc.

2) *Build applications directly from the domain model.* Even if no assets are developed specifically for reuse as a consequence of the domain model, the application development process can benefit from having a consistent language for describing ranges of system capabilities. (Researchers who speak of domain analysis as a step in the application development life cycle that precedes systems analysis may be thinking along these lines).

3) *Use the domain model as an index back to artifacts.* When descriptive modeling has been thorough enough, the domain model can serve as a useful index back into a collection of legacy artifacts. This can support an application engineer working from a domain model who wants to utilize legacy artifacts in building desired applications. ODM, when augmented with the ***traceability*** layer, supports this use of the domain model.

4) *Build assets directly from the domain model.* An asset implementor can use the domain model to specify the features of the assets being developed, without the benefit of an asset base architecture. The implementor could also use traceability information, if present, to identify legacy artifacts that can serve as useful prototypes for the desired assets.

5) *Build a system architecture from the domain model.* After producing the domain model, application developers can create new systems in the domain by deriving individual system architectures from information in the domain model. Each architecture created in this way addresses the requirements of only a single application context.

6) *Build a generic architecture from the domain model.* Following domain modeling, a single "generic architecture" or "domain architecture"[1] can be developed. Such architectures utilize architecture representation techniques similar to those used for individual system architectures, but is structured to be usable within a larger class of applications within the domain. Often the generic architecture will be accompanied with variable sets of components engineered to fill specific slots within the architecture; this is the primary means of supporting variability in such an architecture. In this scenario, the asset base will usually impose an architecture that must be adopted as a standard by all applications making use of the assets.

7) *Build a variable architecture from the domain model.* Following domain modeling, a set of architectural variants can be specified that can be used to create a ***variable architecture***, a configurable architecture framework from which a class of specific system architectures can be derived (or ***instantiated***) by asset utilizers. Like the generic architecture, a variable architecture will be accompanied with components engineered to work within the architecture; however, the mechanisms for supporting variability are broader. For example, there can be variability in the overall topology or structure of the architecture, not just in the selection of components with which to populate the architecture.

The ODM process has been designed to address the entire range of alternatives described above: a domain model is developed which can be used for a variety of purposes, including engineering an asset base using a variety of implementation strategies. Depending on how ambitious a particular project's objectives are, certain processes and workproducts in the core ODM life cycle presented in this document may be of lower priority. This is particularly but not exclusively true in the *Engineer Asset Base* phase.

<u>Example</u>: *Exemplar System Genealogy*

The core ODM process recommends that detailed historical relationships between exemplar systems in the domain be documented in the Plan Data Acquisition task, within the Acquire Domain Information sub-phase of Model Domain. The rationale for performing this analysis at this point in the life cycle is that this information is required in order to make a good Representative Systems Selection. As this can be a potentially labor-intensive analysis to perform, however, it is deferred as late as possible so that modelers only survey the historical connections of systems that are strong candidates for inclusion as representatives.

For an organization that has a clear set of systems of interest defined as part of the organization context (e.g., a system maintenance organization with clear responsibility for a given set of legacy

[1]. The quoted phrases above ("domain architecture", "generic architecture") indicate terms used in the approaches to reuse described, but *not* used as technical terms in ODM. For example, use of the term "domain architecture" is generally avoided in this guidebook, in part because common usage of the term conflates meanings it is important to keep distinct in the ODM context.

systems) there may be intrinsic value in a "Systems Genealogy" that details these historical relationships among all the systems. (Note: this is *not* a defined workproduct within the current ODM process model.) In such a context it may make most sense to perform some of the activities required in the *Plan Data Acquisition* task much earlier in the life cycle; in fact, perhaps before determining the DOMAIN SELECTION or even the PROJECT OBJECTIVES. This is a valid and sound tailoring of the ODM process, because the key benefits of the tasks and workproducts are retained, while risks that motivate the core process model structure are addressed through other means. In this example, the key contextual difference is that there is a valid mechanism for scoping the set of systems that will be the subject of historical analysis.

*Example: Selecting objectives and domain in parallel*

The task structure of both the *Set Project Objectives and Scope Domain* sub-phases represent a transition from a *descriptive* to *prescriptive* orientation within the tasks. It is possible to interleave the descriptive and prescriptive tasks of the two sub-phases: i.e., first performing the *Determine Candidate Project Stakeholders* and *Identify Candidate Project Objectives*; then performing *Characterize Domains of Interest* and *Define Selection Criteria*; then deciding on the PROJECT OBJECTIVES and DOMAIN SELECTION in parallel.

This sequence could have advantages in a stable ORGANIZATION CONTEXT, where *Set Project Objectives* focuses on stakeholders for the context, while in *Scope Domain* planners develop a domain-oriented view of the same context. The results of both initial descriptive tasks can be performed once, then reused for subsequent projects in the same context. This approach requires closer interaction between the organization and domain-oriented views, along with a greater willingness to defer closure on specific commitments and selection. A risk of this approach is expenditure of too many resources on an unfocused survey of potential domains.

## 10.3 Key Challenges

This section offers a few concluding remarks about some key challenges to be considered when putting ODM into practice. In particular, three pervasive challenges are highlighted that will surface in all aspects of the project.

### Handling anxiety around deferring decisions

Domain engineering could almost be described as the art of intentionally deferring decisions. The tendency of engineers to “rush to code” is familiar to project managers who have attempted to introduce more systematic software development practices via CASE technology or specific design methodologies. In domain engineering this tendency will re-emerge continually and at higher levels, i.e., the “rush to design”. In fact, the pressure will be intensified since, in many ways, it is only in the *Scope Asset Bas*e sub-phase, in the later *Engineer Asset Base* phase of domain engineering that a real analogy to the systems requirements phase can be found. For much of the earlier part of the process, engineers will feel themselves in uncharted territory, and the temptation will always be to fall back on familiar activities, i.e., designing solutions.

The degree to which technology decisions can and should be deferred will vary for each organization and project. The objective in ODM is not to defer these decisions by habit or fixed rules, but where there is a strategic benefit in doing so. The challenge for each project will be distinguishing those situations where late binding of decisions is appropriate.

One clear symptom to watch for that indicates this issue is operating is when the project team finds its attention continually drawn to a decision that does not appear to fit the project’s place in the domain engineering life cycle. For example, engineers try to select the domain before objectives are defined; to determine the architecture before the domain model is developed, etc. We offer the following rule of thumb as a test to apply in these cases:

- Does the decision in question actually point to a project constraint?

  If so, it is legitimate for this decision to make the decision early in the process—in fact, as early as possible. If it is a real constraint, it is external to the direct control of the project and therefore not really a decision point at all. There is no point in pretending that the set of customers for the asset base will remain indeterminate until the *Scope Asset Base* sub-phase, if in fact it is known that a certain project must be a customer.

  Document the constraint explicitly in the project constraints. Then move on. The conflict has served as a means for identifying another piece of the "hidden context" for the project. This sort of information belongs in the project constraints from the point it is identified.

- Is there particular risk or uncertainty associated with a particular decision, or a large amount of lead-time required to implement a given decision?

  If this is the case, it may be legitimate to accelerate the decision point. If possible, though, consider using an iterative prototyping strategy. Specify what aspects will be determined by the prototype effort and what will not be determined. Schedule a specific point in the process when the binding decision will be made, to reinforce the fact that the prototype effort is exploratory, or for the purposes of technology evaluation only, etc.

  For example, selection of an ASSET BASE ARCHITECTURE happens very late in the "official" ODM life cycle. In practice, the decision about what architectural approach to take will have numerous repercussions, not only for the project, but for external groups that may be attempting to coordinate their schedules with project results.

If neither of the two cases above seem to apply, consider the possibility that the anxiety and focus around the decision reflects habit, preference, or just greater interest on the part of engineers with the decision in question. If this seems to be part of the problem, then treat deferral of that decision as a specific part of the discipline of domain engineering. The right strategy will not always feel comfortable or familiar. (To be fair, it is also quite possible that the conflict between perceived priorities and the suggested process model indicates an error or context-dependency in the ODM process model itself. Please provide any feedback you can to the method developers to help us to continually refine and improve the method.)

### Dealing with complexity and formality in the process

Even with recent restructuring to separate core aspects of the method from supporting and optional elements, the ODM method is complex, and advocates an unusual degree of formality in both representations and processes followed. This will present a significant technology transfer challenge in most settings. A few simple guidelines may help to reduce the burden this places on those learning and applying ODM:

- <u>Adopt formality incrementally, and only as required</u>. In most cases, it is possible to iterate back to previous steps and add information, increase formality, or establish linkages that were previously left informal. This may require more work than doing it the first time; but if the motivation for the extra formality is not clear at the time it is done, the results may not be worth the effort expended anyway. Formality is a technique to use when the goal is clear.

- <u>Use support technology wherever possible</u>. Employ any and all techniques to reduce the burden of the complexity on the individual engineer. The domain engineering situation by its nature introduces complexity on many simultaneous levels; human beings are generally not good at handling this complexity unassisted. Improving support technology is one of our primary areas of research for the future. Some of the formality suggested in this Guidebook will not be achievable or sustainable without this extra level of automated support. Early adopters of the method will need to be willing to do some seat-of-the-pants domain engineering.

- Learn to enjoy the complexity. Given the two caveats above, the final suggestion is to recognize resistance to complexity when it surfaces, and to counter it with an intentional emphasis on the challenge of learning to manage greater levels of complexity and formality when engaged in the service of broader business and technical goals. Once again, if gratuitous complexity has been introduced in the method, we welcome any feedback to simplify it and make it more accessible. But to the extent that domain engineering does add a level of complexity beyond single-system engineering, enjoy it.

### Integrating diverse skills

The ODM method places particular demands on domain engineering because it requires closely integrated use of skills and disciplines often associated with different staff positions and roles. Some of the specific skill areas required have been organized in terms of the supporting sethods and optional layers discussed in Sections 8 and 9. Integrating these skills is challenging because some skills cross historical boundaries between organizational divisions (e.g., engineering groups vs. marketing vs. management) or even cultural and professional barriers (e.g., organizational and social scientists vs. technical engineers). Furthermore, different aspects and phases of the method will resonate with people of different personal styles and temperaments. While these profiles should be useful in assessing candidate personnel for domain engineering projects, it will be rare to find individuals with strong skills across all disciplines. These guidelines may help in this regard:

- Use team modeling and learning strategies. ODM projects are best performed in the context of multi-disciplinary teams, working collaboratively on planning, information gathering and modeling, and asset base engineering activities. Be alert to the potential of staff from non-engineering backgrounds, such as corporate librarians, technical writers, research assistants, market researchers, and even those traditionally relegated to administrative, clerical or support positions. There are portions of many activities included in the overall domain engineering process that can be effectively delegate to people with these varied backgrounds, given proper training and inclusion in the team.

- Don't skimp on training. Domain engineering is more than a set of formal processes, notations and techniques. It involves developing specific perceptual abilities (e.g., taxonomic modeling skills, perceiving analogies, clustering), increased contextual awareness, and other capacities beyond purely technical skills. No person's particular background provides a guarantee that they will make the transition to domain engineering without training, guidance and team support in the process.

- Treat domain engineering as a personal discipline. Teaming and training strategies aside, each individual can use the domain engineering process as an opportunity to develop personal mastery in unfamiliar areas [Seng90, Seng94, Simo91a]. We believe many of the skills required in domain engineering — separation of descriptive and prescriptive processes, contextual thinking, boundary-setting — will transfer to useful general skills and disciplines. In the end, any method or technology should be judged not only by its specific results, but also by the affect that it has on its practitioners. We hope the influence of ODM as an engineering, conceptual, and social discipline will be both beneficial and enjoyable.

# Appendix A: ODM Process Model

This appendix presents the ODM life cycle modeled using the Process Tree and IDEF$_0$ notations. The full Process Tree diagram and the set of IDEF$_0$ diagrams appear on the pages that follow. The order in which the IDEF$_0$ diagrams appear is consistent with the hierarchical process decomposition structure of ODM and should be straightforward to follow.

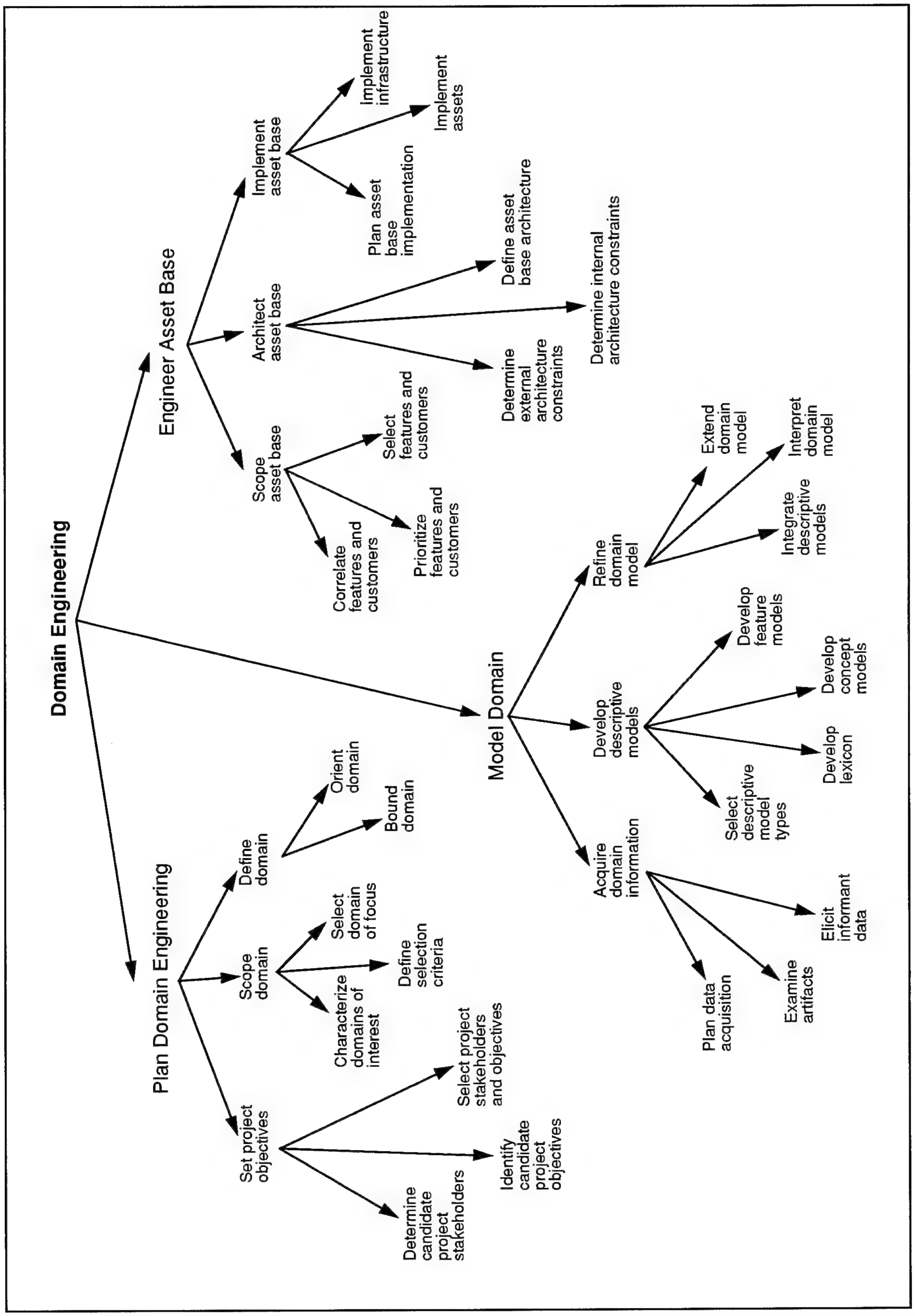
Organizations can use the Process Tree and IDEF$_0$ model directly as a basis for modeling ODM in more detail. They can also adapt the processes and information availability to address their specific needs, or they can integrate the model (or some adaptation thereof) with existing IDEF$_0$ process models.

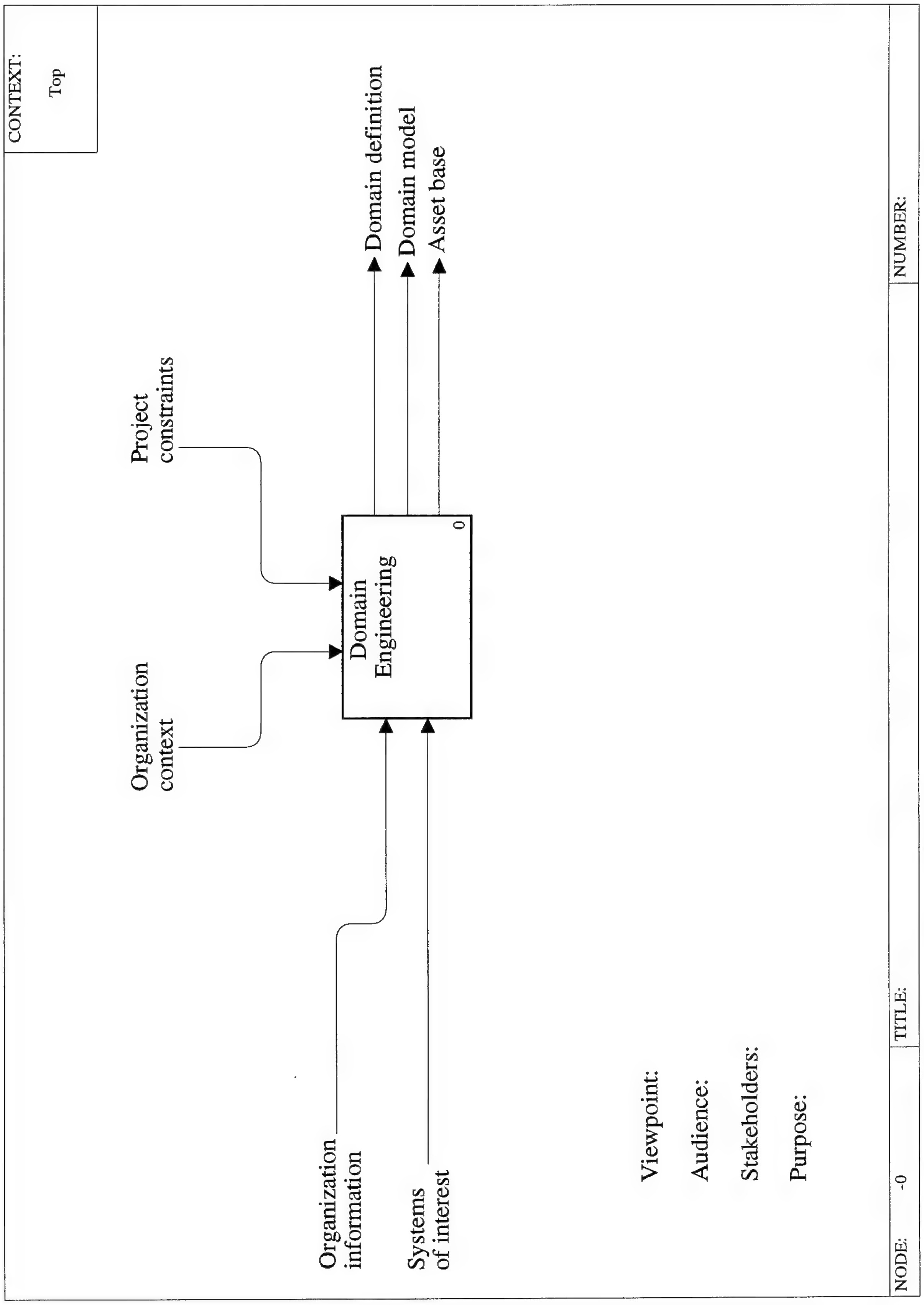
Process Trees were developed on the Army/Unisys STARS Demonstration Project to depict multiple levels of functional decomposition of a process in a hierarchical, graphical representation. A Process Tree decomposition appears as activities connected by arrows. Arrows decompose an activity to its subactivities. Process Trees provide a convenient way of navigating complex process decompositions. Nodes within Process Trees are used to index into the ODM IDEF$_0$ Diagrams.

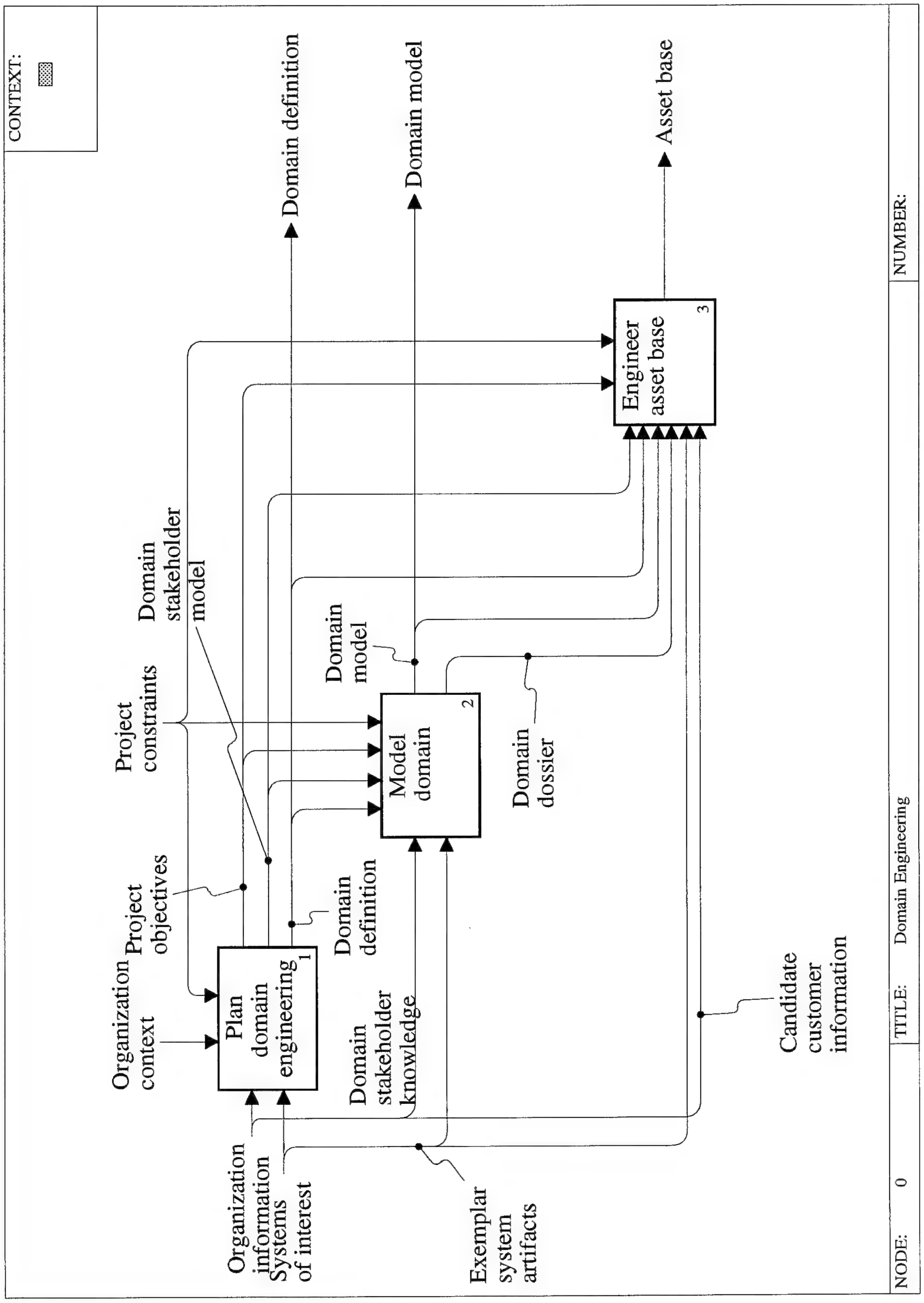
IDEF$_0$ is becoming an increasingly popular process modeling notation, but not all readers of this document may be familiar with it. The following is a brief overview of the IDEF$_0$ notation:

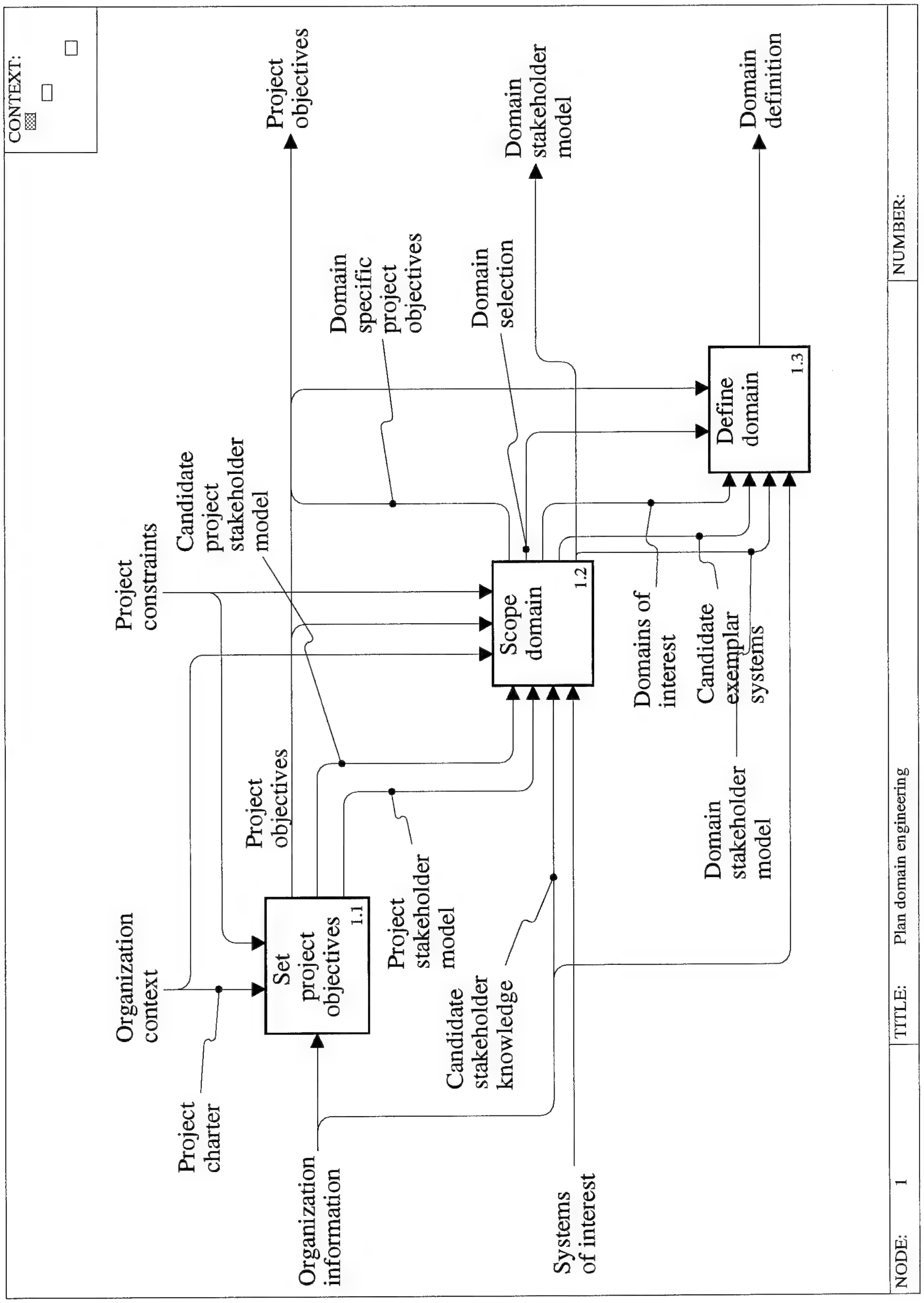
> An IDEF$_0$ activity diagram contains one level of decomposition of a process. Boxes within a diagram show the subactivities of the parent activity named by the diagram. Arrows between the boxes show the availability of information to activities. Arrows entering the left side of a box are inputs to an activity, arrows exiting the right side of a box are outputs from the activity, arrows entering the top of a box are controls that regulate the activity, and arrows entering the bottom of a box are mechanisms that support the activity. A sequential ordering of boxes in a diagram does not imply a sequential flow of control between the activities. Any activity can be further decomposed into another IDEF$_0$ diagram describing its subactivities.

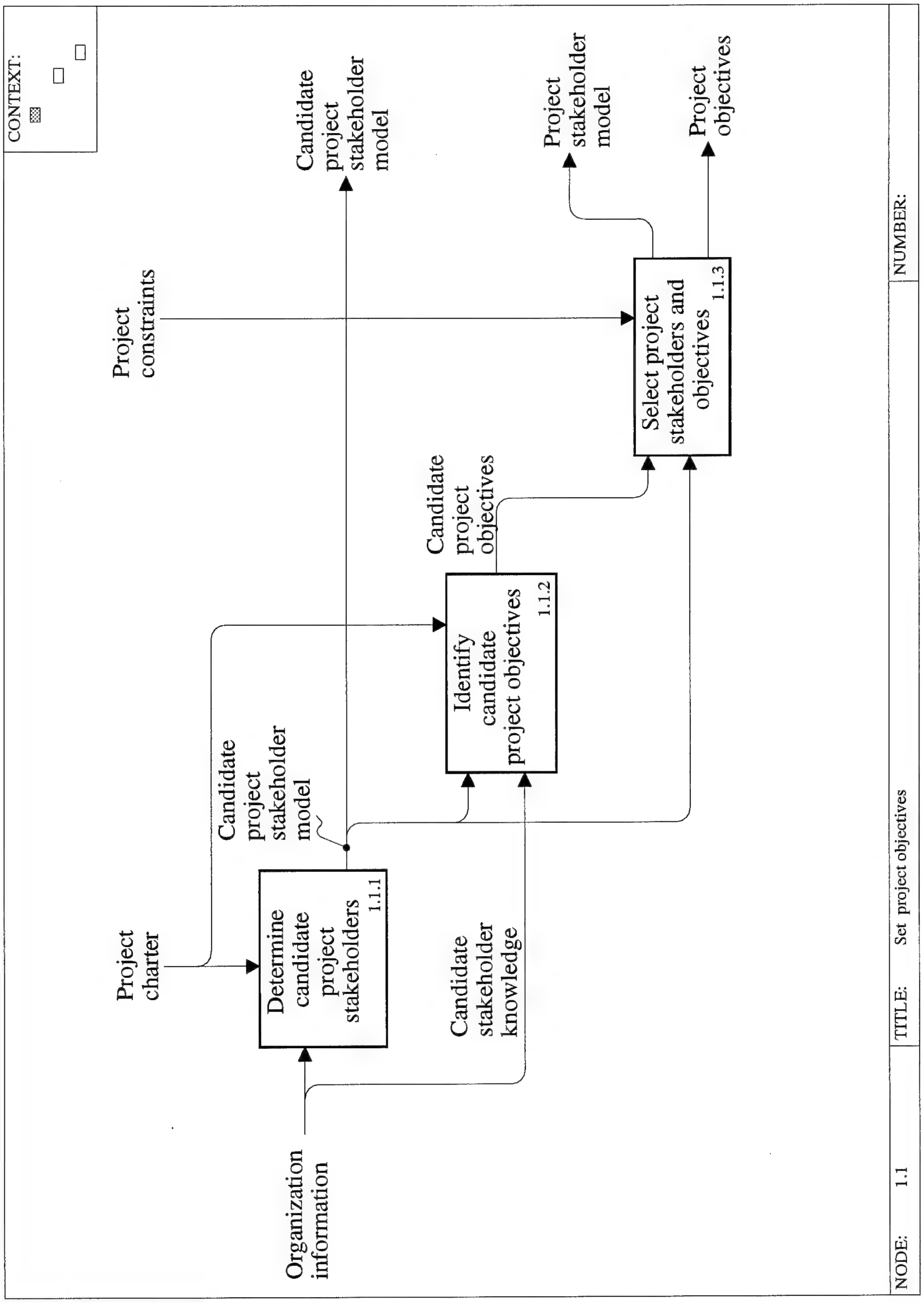
See [IDEF81, Marc88] for more details on the notation syntax and semantics.

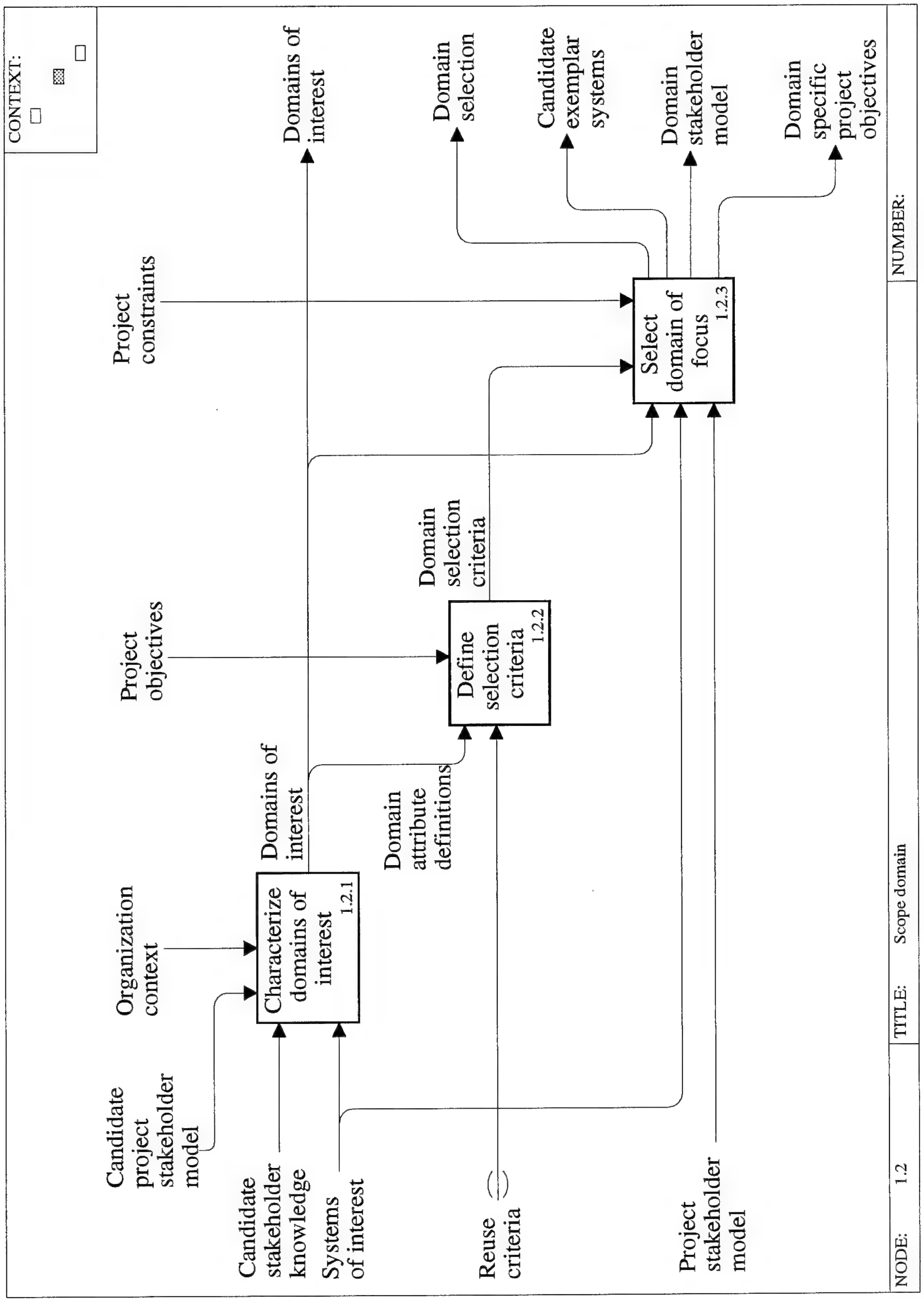

Domain Engineering
Plan Domain Engineering
Set project objectives
Determine candidate project stakeholders
Identify candidate project objectives
Select project stakeholders and objectives
Scope domain
Characterize domains of interest
Define selection criteria
Select domain of focus
Define domain
Bound domain
Orient domain
Model Domain
Acquire domain information
Plan data acquisition
Examine artifacts
Elicit informant data
Develop descriptive models
Select descriptive model types
Develop lexicon
Develop concept models
Develop feature models
Refine domain model
Integrate descriptive models
Interpret domain model
Extend domain model
Engineer Asset Base
Scope asset base
Correlate features and customers
Prioritize features and customers
Select features and customers
Architect asset base
Determine external architecture constraints
Determine internal architecture constraints
Define asset base architecture
Implement asset base
Plan asset base implementation
Implement assets
Implement infrastructure


CONTEXT:
Top
Organization context
Project constraints
Domain Engineering
0
Domain definition
Domain model
Asset base
Organization information
Systems of interest
Viewpoint:
Audience:
Stakeholders:
Purpose:
NODE:
-0
TITLE:
NUMBER:

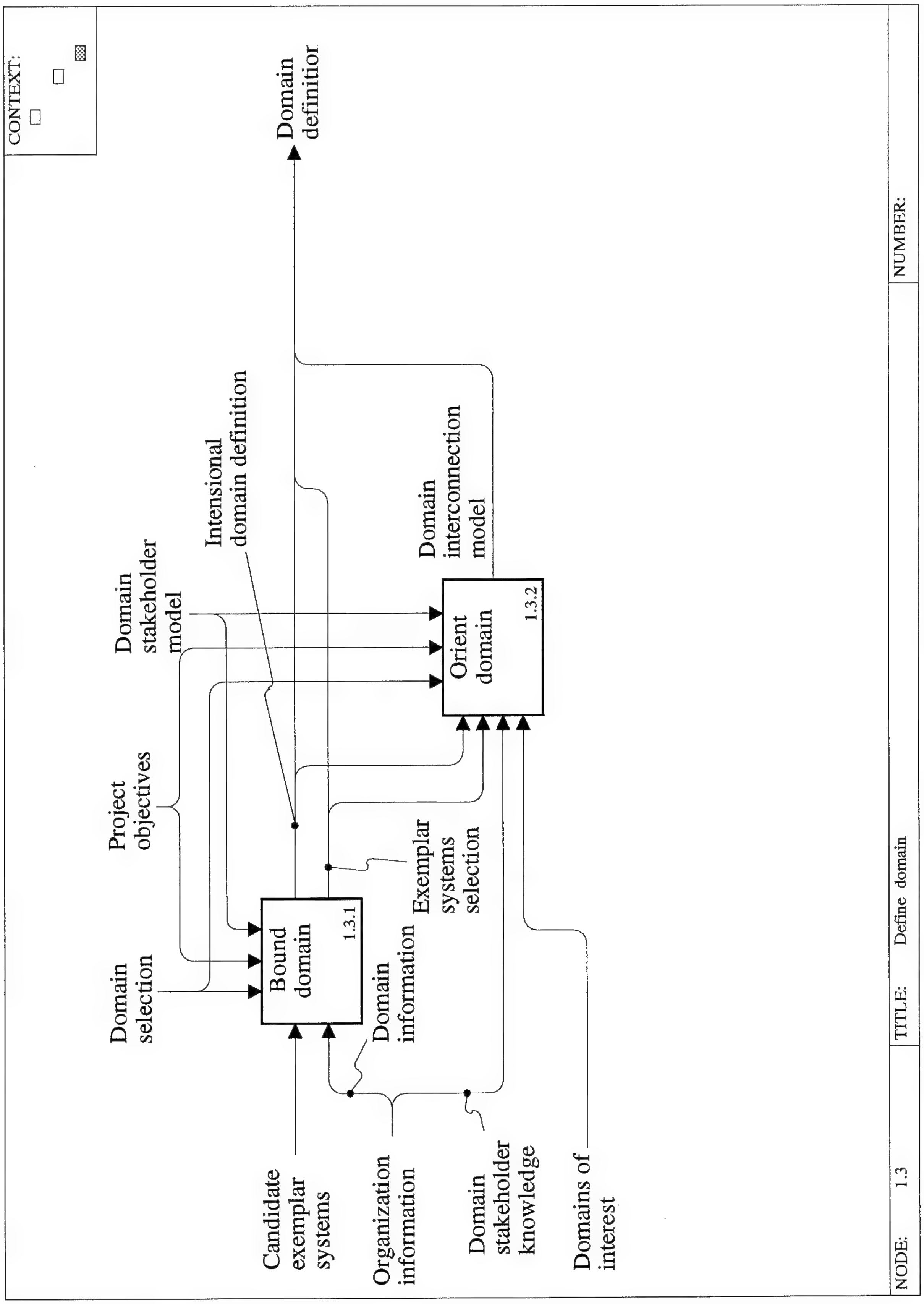

CONTEXT:
Plan domain engineering 1
Model domain 2
Engineer asset base 3
Organization context
Project objectives
Project constraints
Domain stakeholder model
Domain definition
Domain model
Domain dossier
Asset base
Domain stakeholder knowledge
Organization information
Systems of interest
Exemplar system artifacts
Candidate customer information
NODE: 0
TITLE: Domain Engineering
NUMBER:

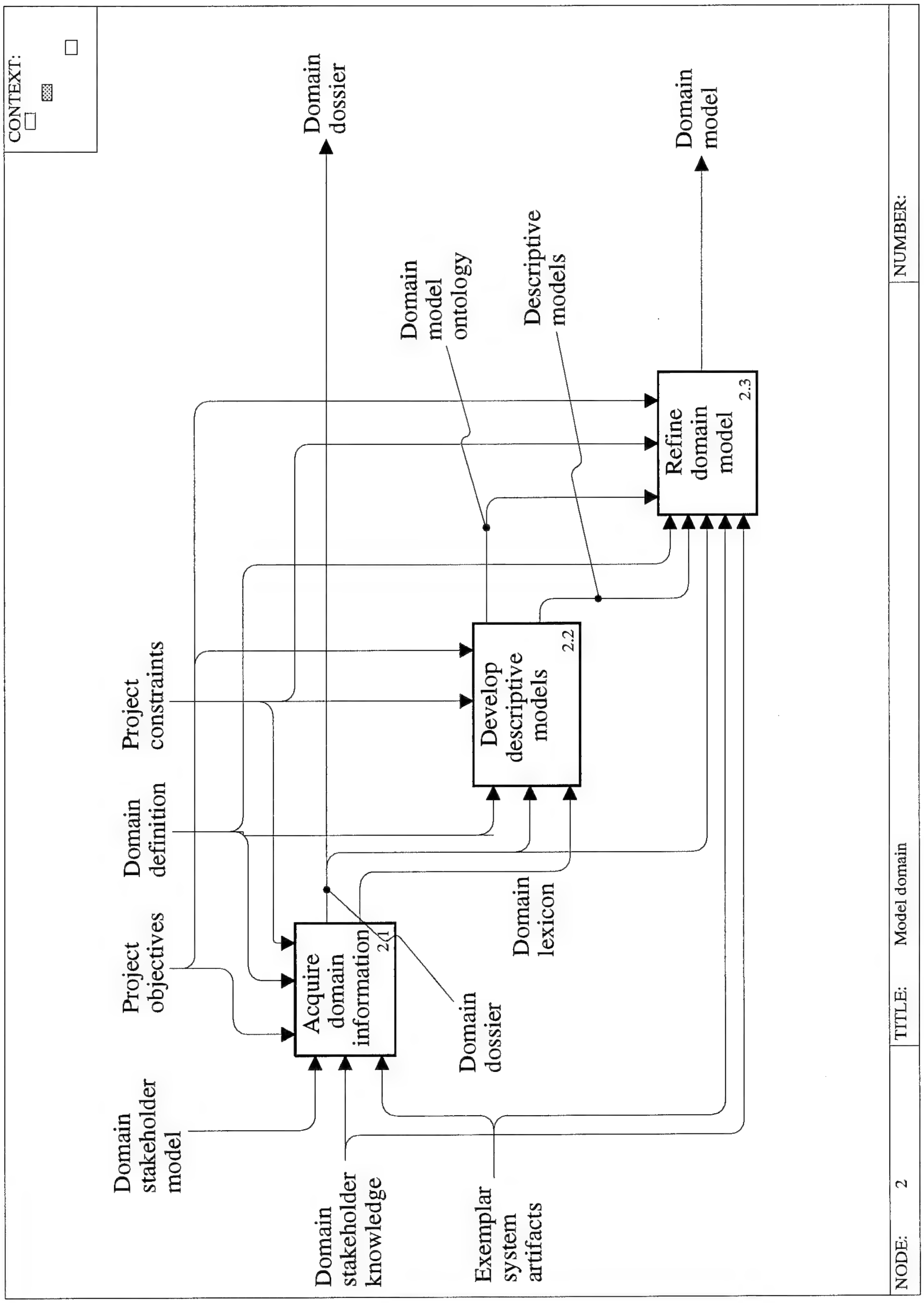

CONTEXT:
Set project objectives
1.1
Scope domain
1.2
Define domain
1.3
Organization context
Project charter
Organization information
Project constraints
Candidate project stakeholder model
Project objectives
Project stakeholder model
Candidate stakeholder knowledge
Systems of interest
Domain specific project objectives
Domain selection
Domains of interest
Candidate exemplar systems
Domain stakeholder model
Project objectives
Domain stakeholder model
Domain definition
NODE:
1
TITLE:
Plan domain engineering
NUMBER:

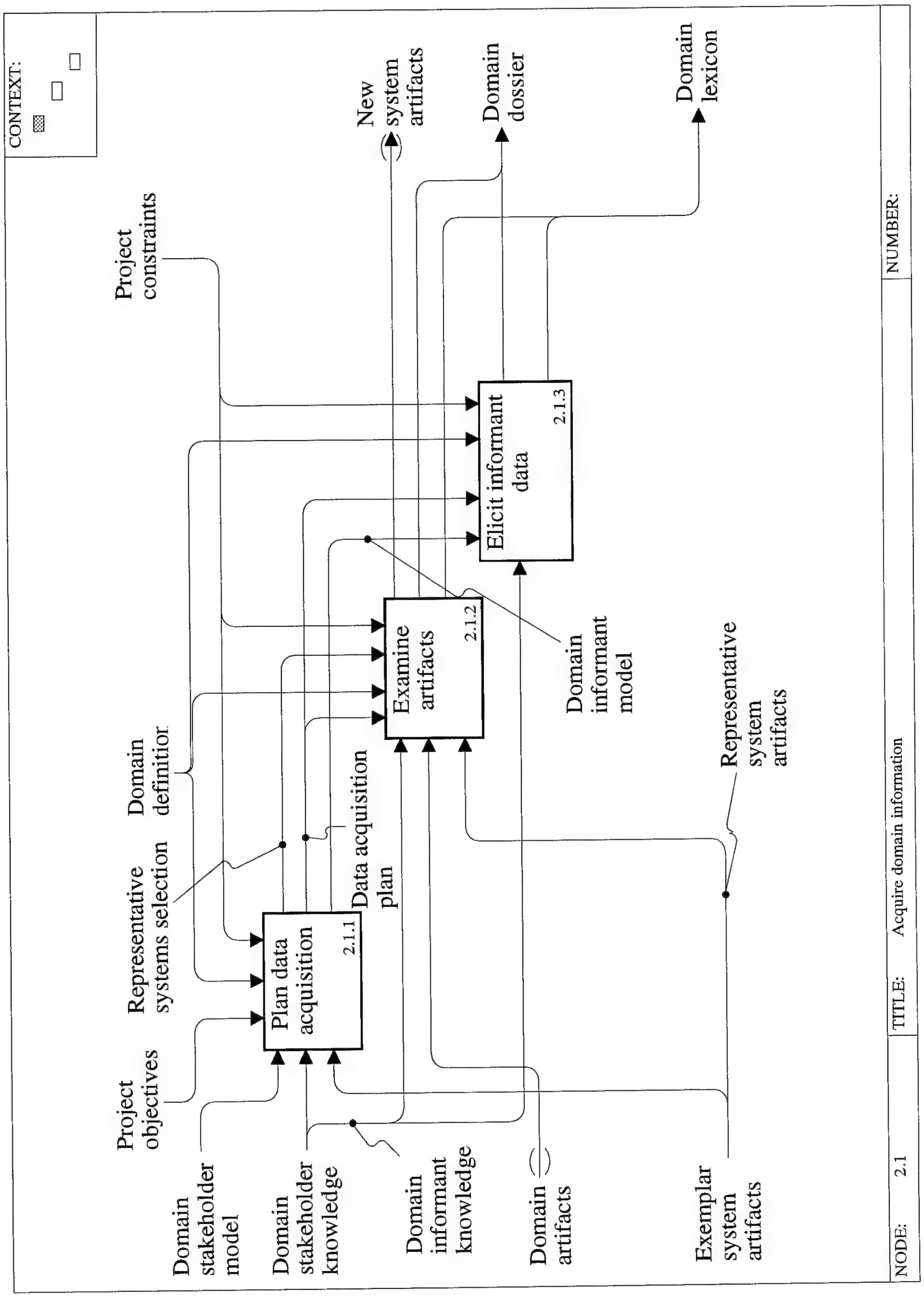

CONTEXT:
Project charter
Project constraints
Organization information
Determine candidate project stakeholders 1.1.1
Candidate project stakeholder model
Candidate stakeholder knowledge
Identify candidate project objectives 1.1.2
Candidate project objectives
Select project stakeholders and objectives 1.1.3
Candidate project stakeholder model
Project stakeholder model
Project objectives
NODE: 1.1
TITLE: Set project objectives
NUMBER:

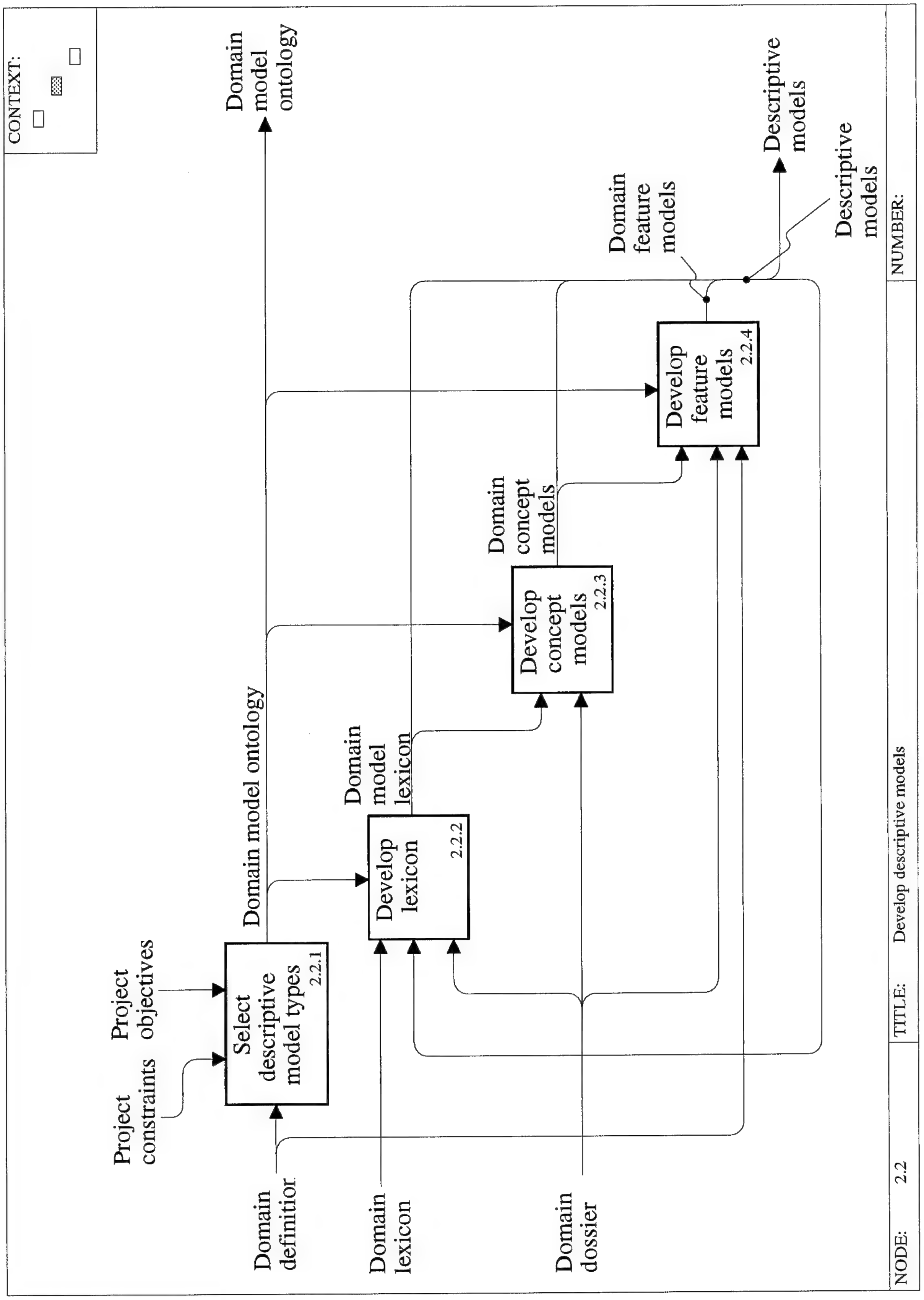

CONTEXT:
Candidate project stakeholder model
Organization context
Project objectives
Project constraints
Candidate stakeholder knowledge
Systems of interest
Reuse criteria
Project stakeholder model
Characterize domains of interest 1.2.1
Domains of interest
Domain attribute definitions
Define selection criteria 1.2.2
Domain selection criteria
Select domain of focus 1.2.3
Domains of interest
Domain selection
Candidate exemplar systems
Domain stakeholder model
Domain specific project objectives
NODE: 1.2
TITLE: Scope domain
NUMBER:

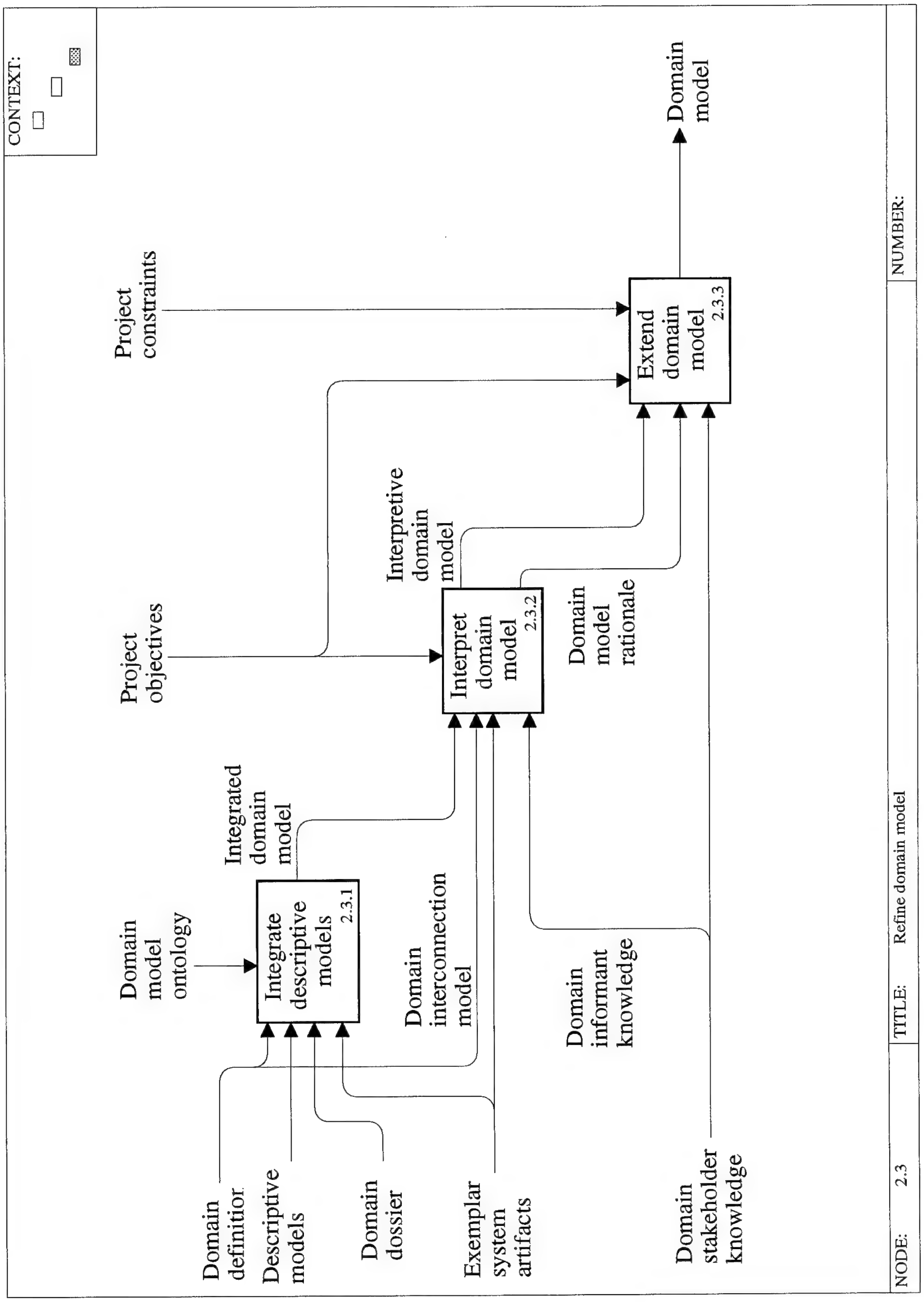
]

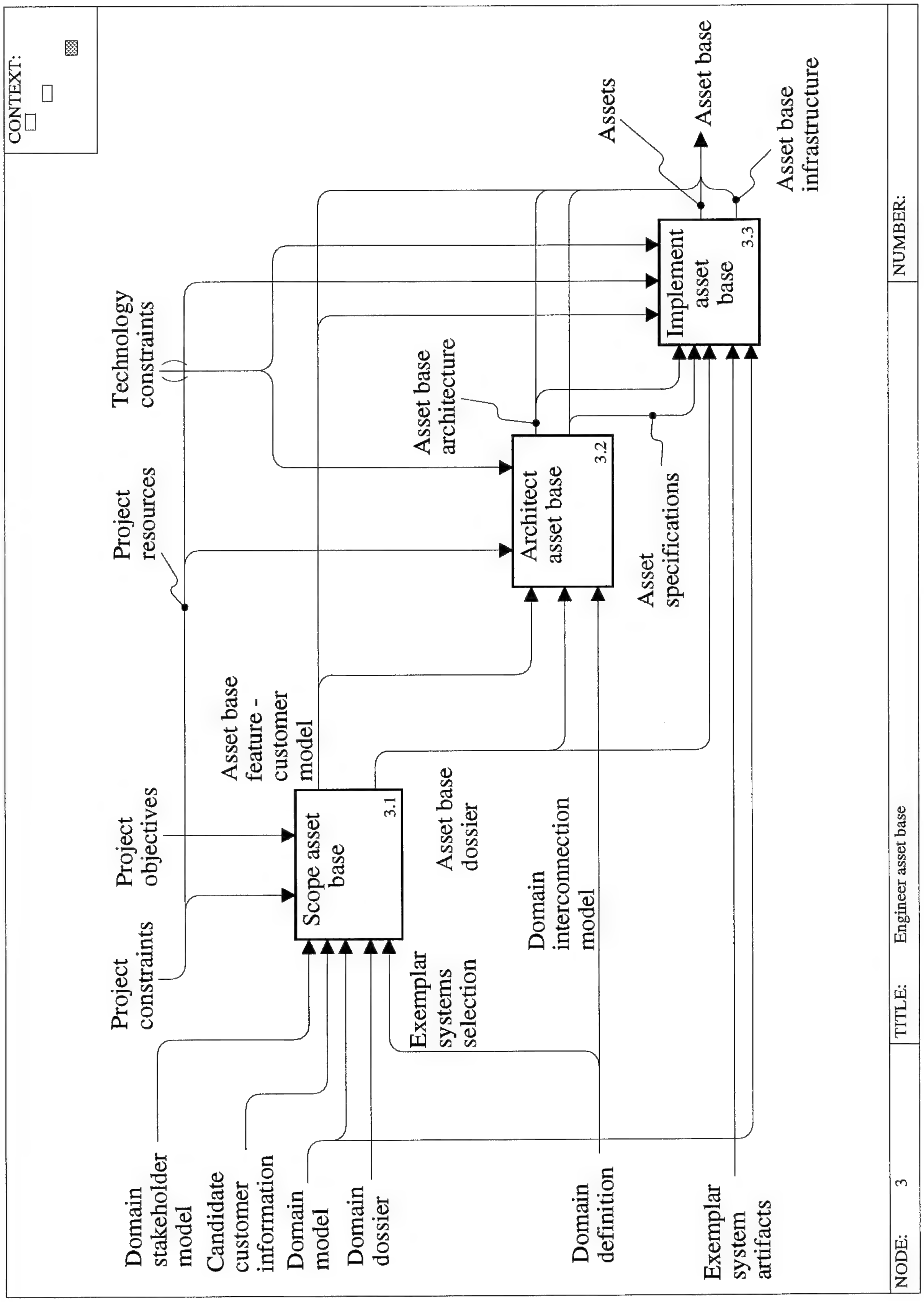

CONTEXT:
Domain definitior
Intensional domain definition
Domain stakeholder model
Domain interconnection model
Project objectives
Orient domain 1.3.2
Domain selection
Bound domain 1.3.1
Exemplar systems selection
Domain information
Candidate exemplar systems
Organization information
Domain stakeholder knowledge
Domains of interest
NODE: 1.3
TITLE: Define domain
NUMBER:

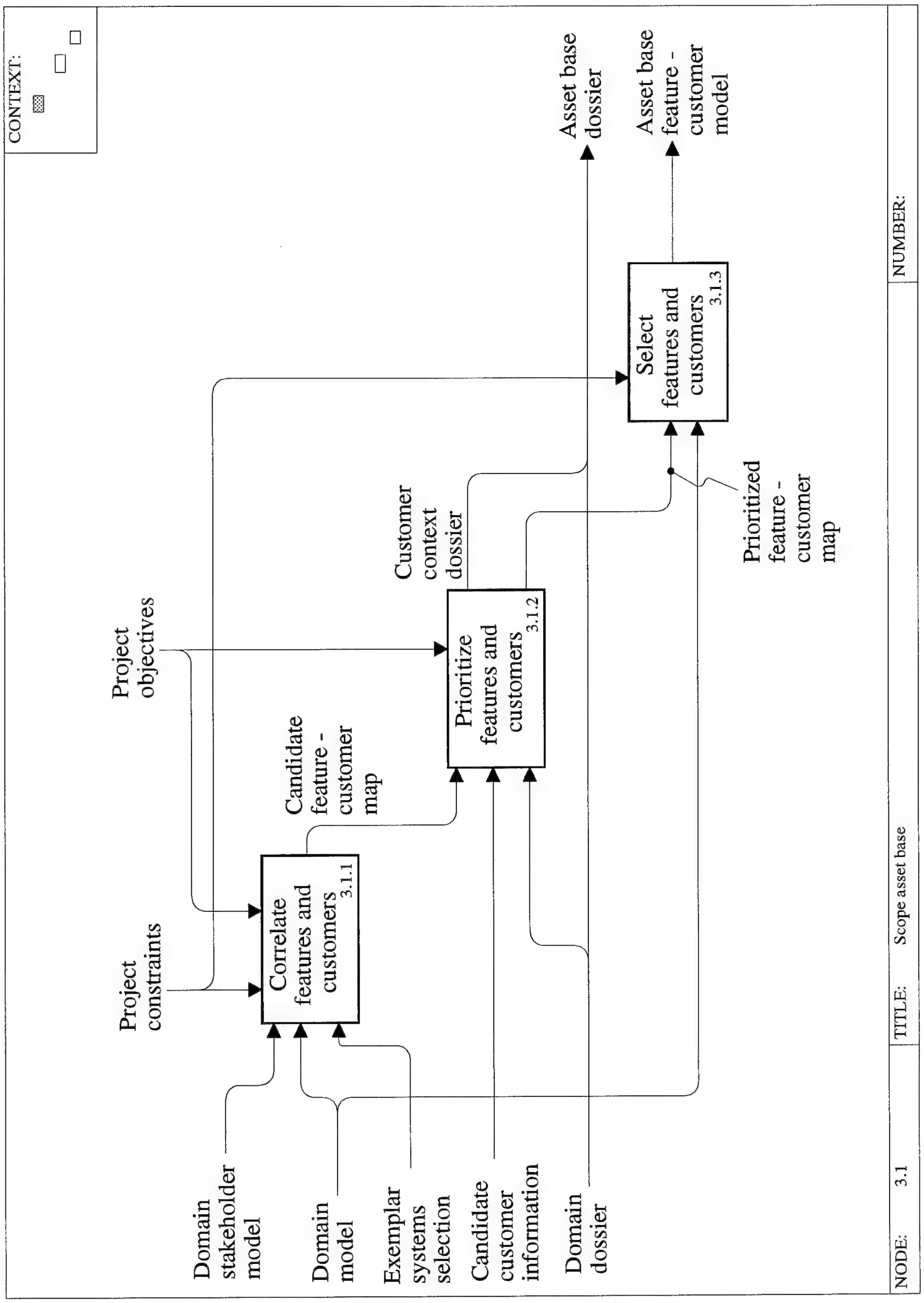

CONTEXT:
Domain dossier
Domain model
Domain model ontology
Descriptive models
Refine domain model
2.3
Develop descriptive models
2.2
Acquire domain information
2.1
Project constraints
Domain definition
Project objectives
Domain lexicon
Domain dossier
Domain stakeholder model
Domain stakeholder knowledge
Exemplar system artifacts
NODE: 2
TITLE: Model domain
NUMBER:

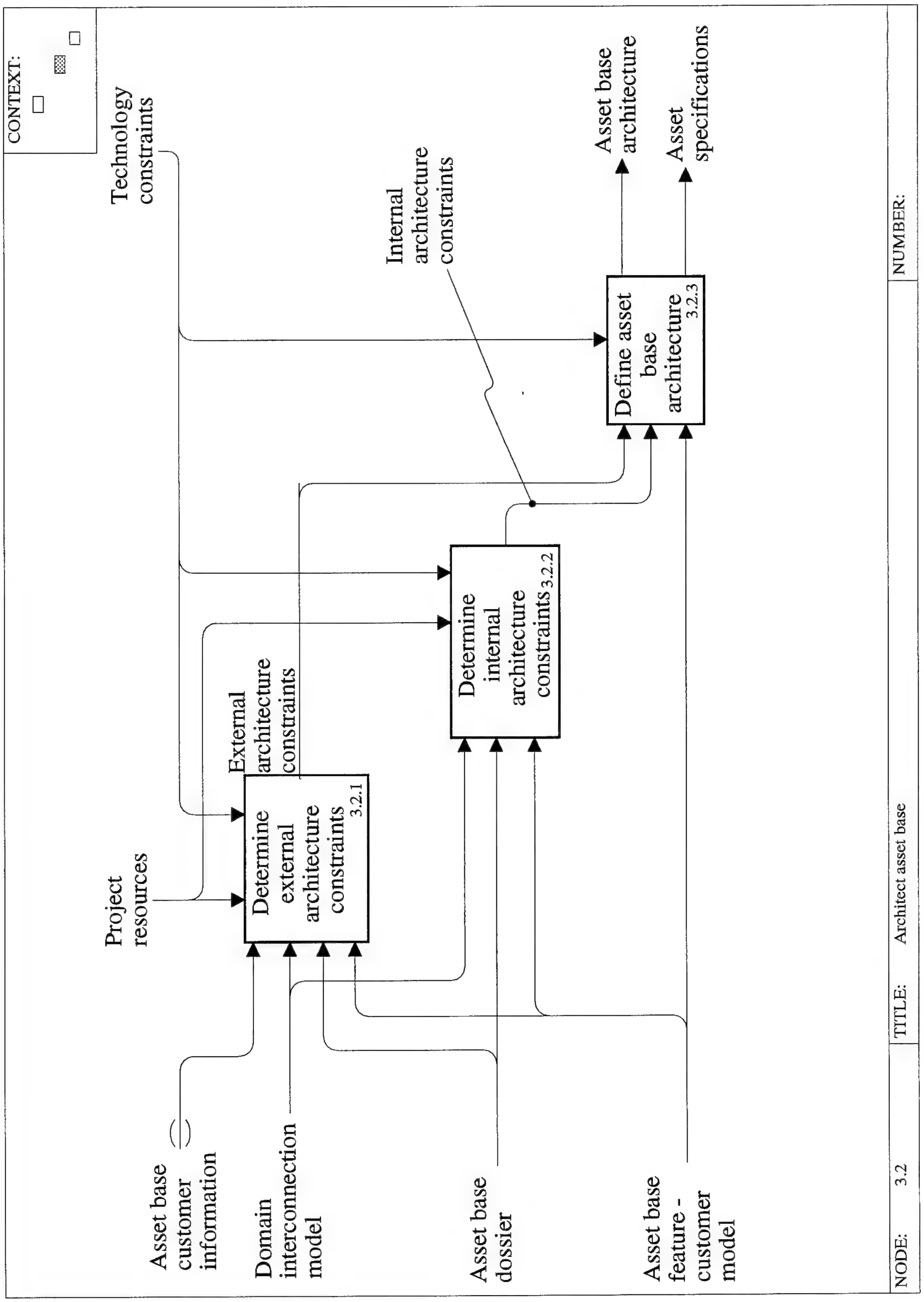

CONTEXT:
Project constraints
New system artifacts
Domain dossier
Domain lexicon
Domain definition
Representative systems selection
Project objectives
Plan data acquisition
2.1.1
Examine artifacts
2.1.2
Elicit informant data
2.1.3
Data acquisition plan
Domain informant model
Representative system artifacts
Domain stakeholder model
Domain stakeholder knowledge
Domain informant knowledge
Domain artifacts
Exemplar system artifacts
NODE: 2.1
TITLE: Acquire domain information
NUMBER:

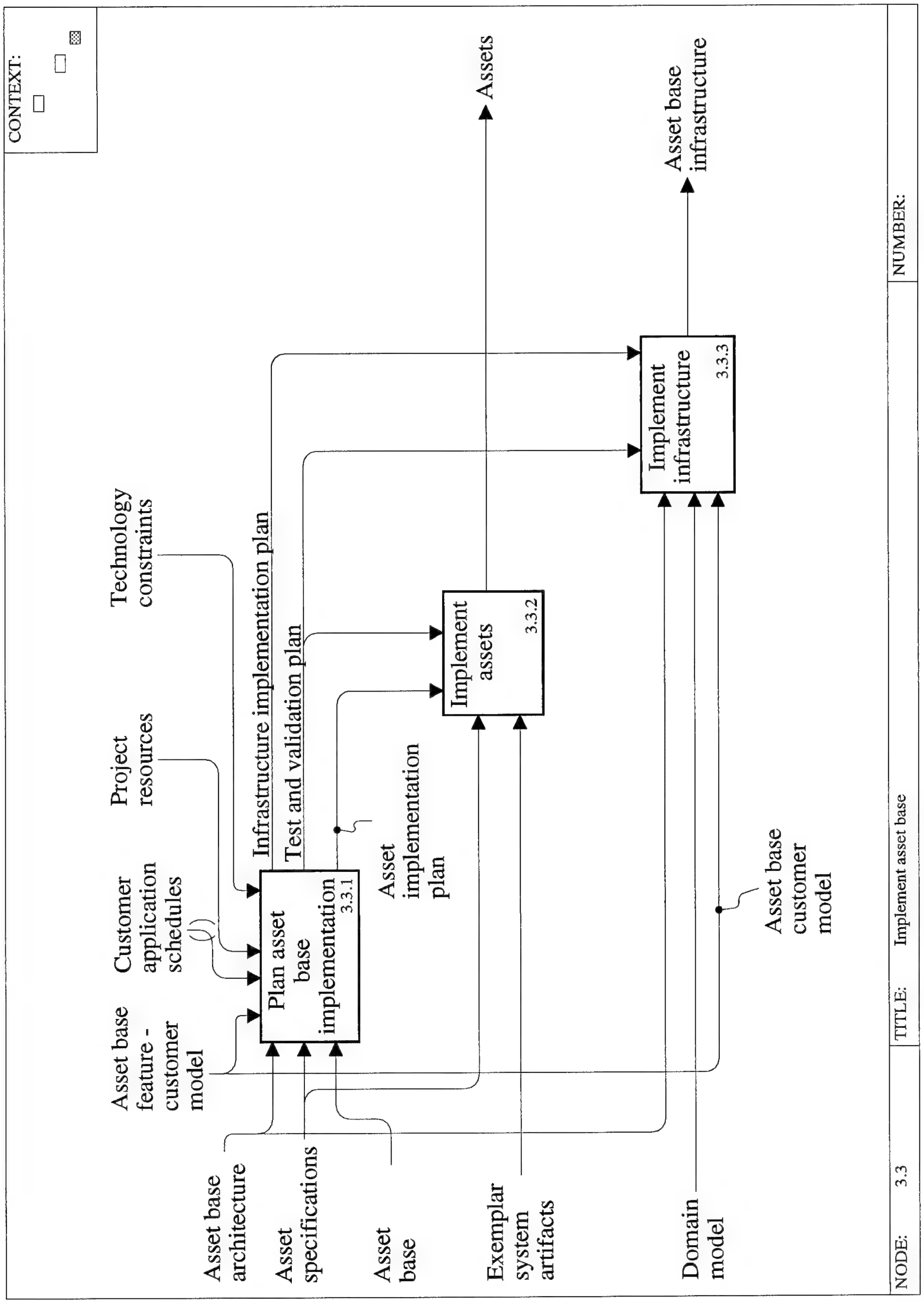

CONTEXT:
Project constraints
Project objectives
Domain definition
Domain lexicon
Domain dossier
Select descriptive model types 2.2.1
Domain model ontology
Develop lexicon 2.2.2
Domain model lexicon
Develop concept models 2.2.3
Domain concept models
Develop feature models 2.2.4
Domain feature models
Domain model ontology
Descriptive models
Descriptive models
NODE: 2.2
TITLE: Develop descriptive models
NUMBER:


CONTEXT:
Domain model ontology
Project objectives
Project constraints
Integrate descriptive models 2.3.1
Interpret domain model 2.3.2
Extend domain model 2.3.3
Integrated domain model
Interpretive domain model
Domain model rationale
Domain model
Domain definition
Descriptive models
Domain dossier
Exemplar system artifacts
Domain stakeholder knowledge
Domain interconnection model
Domain informant knowledge
NODE: 2.3
TITLE: Refine domain model
NUMBER:


CONTEXT:
Project constraints
Project objectives
Project resources
Technology constraints
Domain stakeholder model
Candidate customer information
Domain model
Domain dossier
Domain definition
Exemplar system artifacts
Scope asset base
3.1
Asset base feature - customer model
Asset base dossier
Exemplar systems selection
Domain interconnection model
Architect asset base
3.2
Asset base architecture
Asset specifications
Implement asset base
3.3
Assets
Asset base
Asset base infrastructure
NODE: 3
TITLE: Engineer asset base
NUMBER:


CONTEXT:
Domain stakeholder model
Domain model
Exemplar systems selection
Candidate customer information
Domain dossier
Project constraints
Project objectives
Correlate features and customers 3.1.1
Candidate feature - customer map
Prioritize features and customers 3.1.2
Customer context dossier
Select features and customers 3.1.3
Prioritized feature - customer map
Asset base dossier
Asset base feature - customer model
NODE: 3.1
TITLE: Scope asset base
NUMBER:


CONTEXT:
Technology constraints
Internal architecture constraints
Asset base architecture
Asset specifications
Define asset base architecture 3.2.3
Determine internal architecture constraints 3.2.2
External architecture constraints
Determine external architecture constraints 3.2.1
Project resources
Asset base customer information
Domain interconnection model
Asset base dossier
Asset base feature - customer model
NODE: 3.2
TITLE: Architect asset base
NUMBER:


CONTEXT:
Asset base feature - customer model
Customer application schedules
Project resources
Technology constraints
Infrastructure implementation plan
Test and validation plan
Plan asset base implementation 3.3.1
Asset implementation plan
Implement assets 3.3.2
Assets
Implement infrastructure 3.3.3
Asset base infrastructure
Asset base customer model
Asset base architecture
Asset specifications
Asset base
Exemplar system artifacts
Domain model
NODE: 3.3
TITLE: Implement asset base
NUMBER:

# Appendix B: ODM Lexicon

## Conventions

This lexicon defines terms used in the document. Three types of terms are included in the lexicon:

- General descriptive terms for concepts essential to ODM appear in the lexicon in **lower-case bold type**. Within the document, these terms when first referenced in any given section appear in ***bold italic*** type.

  We have tried to include only terms that are used with special meanings in the ODM context. Some of these terms are used frequently in other areas of software engineering (e.g., "system") or general usage (e.g., "organization"). They are used in a specific technical sense in ODM, i.e., to serve as an umbrella for several more specific terms, or to make key distinctions that would otherwise be difficult to convey. For example, the term "asset" is used in preference to the more familiar term "component" to denote a product of domain engineering that can take the form of a component or a generator. This usage emphasizes that the implementation technology used can vary from asset to asset. If you see a term in ***bold italic***, make sure you are clear about the specific meaning it has in the ODM context.

- Workproduct names appear in the lexicon in **BOLD SMALL CAPITALS**, with initial large caps. These terms are provided here as a cross-reference into Appendix C.1, Workproducts Definitions. There are three types of entries for workproducts:

  — *Main* workproducts are top-level outputs listed in the Workproducts subsection of the various task descriptions in the document. They also appear as outputs on the $IDEF_0$ diagrams that document the process model throughout the document and in Appendix A. In the lexicon, these workproducts are directly cross-referenced to entries under the same name in Appendix C.1, which is listed alphabetically by *main workproduct name*.

  — *Component* workproducts are those that appear listed hierarchically under main workproducts in the task descriptions. They do *not* appear on the $IDEF_0$ diagrams. In the lexicon, they are cross-referenced to the *main workproduct* entry of which they are a component. Look up the definition directly in Appendix C.1 under the main workproduct name.

  — *Composite* workproducts are higher-level data flows indicated on the $IDEF_0$ diagrams that aggregate several main workproducts (or other composite workproducts). These appear in this lexicon as direct cross-references to Appendix C.1, like main workproducts. In Appendix C.1, however, they are listed as composite workproducts with their constituent workproduct elements and not further defined.

- Data flow names correspond to net inputs to the ODM process as described in the $IDEF_0$ diagrams. They appear in the lexicon in **BOLD SMALL CAPITALS**. Unlike the workproducts they are defined directly in the lexicon. Composite and component data flows are also defined in the lexicon.

- Process names and workproduct names within the body of lexicon term definitions follow the same conventions as in the main part of the document. Phase, sub-phase, and task names are in *Lower Case Italic* with initial large caps. Workproduct names are in Small Capitals, with initial large caps. Process names are not defined in the lexicon, since they can be found in Part II.

- Formal and informal terms. In many cases a term like "informant" is used extensively throughout the document, when in fact the meaning of the term is "[domain] informant." In these cases, we have chosen to place the main definition at the *less* formal term, since that is the most common usage in the document.

**accessibility**

An evaluation criterion for informants in the *Acquire Domain Information* sub-phase. Describes the ease of access to the informant, based on factors such as their commitment to application schedules, place in the organization, etc.

**adequate**

A DOMAIN FEATURE MODEL is adequate with respect to the REPRESENTATIVE SYSTEMS SELECTION when there is a feature variant within the model that corresponds to the distinct characteristics of each representative system with respect to that feature category.

**ad hoc reuse**

Use of a software workproduct in a context for which it was not originally developed, without an intermediary step of reengineering the artifact into an asset, i.e., for reuse in multiple contexts.

**agent**

A person, tool, or executing application program. Agents fill roles to perform work. (See also practitioner)

**analogue**

a relation between two or more exemplar artifacts that perform similar functions within their respective system contexts, and have been structured for comparison during the *Examine Artifacts* task. Can also be applied to terms in the DOMAIN MODEL LEXICON that are part of the same term cluster.

**analogy domain**

Formally, two domains D1 and D2 are said to be analogous if they have overlapping sets of associated exemplar systems or defining features, with neither being a complete subset of the other. Informally, an analogy domain is one related to the domain of focus through historical or cognitive associations by practitioners, e.g., the family tree domain is analogous to the Outlining domain.

**architectural feature**

A distinctive feature of a system architecture. The feature could be a global attribute, e.g., "is client-server", or a specific structural component of the architecture; e.g., a specific layer of a system with a layered architecture topology could be termed a structural feature of that architecture. If a different exemplar system did not separate out that layer, this would become a basis for structural comparison of the two architectures.

The distinction between an architectural feature and a design feature is not defined within ODM. This distinction would depend on the particular choice of methodology for system design and/or system architecture used by the project. The important point is the recognition that feature analysis can be approached from a top-down or architectural perspective.

### architectural variant

An architectural description that differs from another architecture variant with respect to some architectural features. For example, if one exemplar system was client-server based and another was not, each could be an architectural variant with respect to the architectural feature "encapsulation strategy".

### architecture

Within the context of ODM, an architecture refers to a structure for inter-connecting a set of constituent components. The components can be workproducts from any point in the software life cycle. Thus, it is legitimate to speak of a **requirements architecture**, meaning a collection of requirements organized into an overall structure (e.g., a requirements document); a **test-case architecture**, denoting an arrangement of individual test cases according to some organizing principle appropriate to the testing methodology in use, etc. The conventional notion of an architecture familiar to most readers would be a **design** and/or **implementation architecture**, that is, an arrangement of software design workproducts or code modules according to an overall structuring principle.

The term architecture is usually applied to entire systems. In the ODM context, where the domain of focus may consist of a sub-system scope of functionality, the term **architecture** can be applied in a relative sense to denote a structuring principle spanning the entire sub-system in the domain context. This means that architectural variants produced during Asset Base Engineering will be architectures for the domain scope, not necessarily architectures for entire systems within which the domain functionality occurs. The interface between domain functionality and the surrounding system context will usually be addressed as part of an architectural variant.

With these exceptions, ODM's notion of an architecture is compatible with most current work in the area of software architectures. These sources should be consulted for more detailed understanding of architecture issues, such as [Shaw94a, Shaw95a, Clem95a, Perr92a].

### architecture-centric

An approach to reuse that emphasizes the development of asset bases organized around specific domains and supporting reuse of entire system architectures and components within those architectures.

### architecture-driven

A top-down approach to the selection of prescriptive features by exploring feature sets describing the domain system as a whole. See also feature-driven.

### architecture feature model

A feature model differentiating exemplars on the basis of architectural features.

### architecture styles

Researchers in software architectures have recognized dramatic structural similarities in the architectures of systems built to perform diverse real-world missions, and developed and used by diverse communities. Some of these structural patterns are beginning to be codified and catalogued as architecture styles [Shaw94a, Shaw95a].

### artifact

A legacy system workproduct. Artifacts contain raw data about an exemplar system, such as a code component, a bug report, a requirements document. The legacy system could be a legacy system not currently in use as part of any development process or a system never deployed (canceled prototype, etc.). An artifact is distinct from an asset in that it is not assumed to be under the control of a reuse/asset manager. It is not incorporated into an asset base, and has not been specifically reengineered for reuse.

The term artifact denotes the fixing of information in actual documentary form (such as on-line textual information). Domain information, e.g., knowledge about the domain in an expert's head or expressed in a conversation, must be fixed into the form of some artifact, e.g., an interview report or transcription, before it can be dealt with concretely in domain engineering. At this point it becomes an artifact.

Artifacts include:

- Primary artifacts which are used as workproducts in creating exemplar systems. These include tools. These are also the subject of direct comparative modeling during the *Develop Descriptive Models* sub-phase, and are typically candidates for reengineering into assets.
- Secondary artifacts, including review articles, commentary, surveys etc. Secondary artifacts can include exemplar-specific artifacts as well as artifacts associated with particular informants but not with one particular exemplar system.

Other artifacts to be considered can include informal or formal domain modeling efforts from domain stakeholders (earlier domain models, architectures, etc.) These must be handled differently from other artifacts because they contain direct taxonomic modeling information. For example, a generic architecture developed by a design team using informal reuse techniques could not be easily compared with a representative system built for a single context of use. Also, artifacts generated by the domain engineering team itself (e.g., via reverse engineering) should be distinguished from primary artifacts.

## artifact-based asset

Artifact-based assets include traditional components that will be retrieved by application developers, possibly adapted or modified, and then incorporated into their own products. A simple example would be a reusable code module, engineered from several analogous code artifacts performing a similar function. Other artifact-based assets could also be engineered from interim artifacts, used in system development but not part of the deliverable system. Examples might include requirements checklists used by analysts as reminders, test data, debugging scripts, etc. See also process asset, stakeholder-based asset.

### ARTIFACTS LIST

See ARTIFACTS LIST under DOMAIN DOSSIER in Appendix C.

## asset

A workproduct developed or reengineered for reuse and placed under management within an asset base. Assets can be developed from workproducts from any phase of the software engineering life cycle, including requirements, design, code components, test cases, etc. Assets can also be reengineered versions of informal engineering artifacts such as checklists, process descriptions, and guidelines.

Assets can include both static components, reused by being adapted and incorporated into deliverable systems, and tools and generators, that are reused by application developers executing them as programs to generate tailored assets. See also artifact-based asset, process-based asset, stakeholder-based asset.

**ASSET BASE**

See ASSET BASE in Appendix C.

**asset base architect**

A domain engineer who defines an ASSET BASE ARCHITECTURE.

**ASSET BASE ARCHITECTURE**

See ASSET BASE ARCHITECTURE in Appendix C.

**asset base customer**

A stakeholder organization, or groups within a system development effort or other application context who are intended utilizers of assets in the asset base being developed. The asset base customers provide the specific target adoption group for the *Engineer Asset Base* phase. (Also referred to as "customer.")

**ASSET BASE CUSTOMER INFORMATION**

A component data flow of CANDIDATE CUSTOMER INFORMATION. Information about asset base customers selected in the *Scope Asset Base* sub-phase, used in the *Determine External Architecture Constraints* task.

**ASSET BASE CUSTOMER MODEL**

See ASSET BASE CUSTOMER MODEL under ASSET BASE MODEL in Appendix C.

**ASSET BASE DOSSIER**

See ASSET BASE DOSSIER in Appendix C.

**asset base engineering**

Scoping and architecting of the ASSET BASE, and implementation of ASSETS and the ASSET BASE INFRASTRUCTURE.

**ASSET BASE FEATURE — CUSTOMER MAP**

See ASSET BASE FEATURE — CUSTOMER under ASSET BASE MODEL in Appendix C.

**ASSET BASE FEATURE MODEL**

See ASSET BASE FEATURE MODEL under ASSET BASE MODEL in Appendix C.

**ASSET BASE INFRASTRUCTURE**

See ASSET BASE INFRASTRUCTURE in Appendix C.

**ASSET BASE MODEL**

See ASSET BASE MODEL in Appendix C.

**ASSET DELIVERY MECHANISM**

See ASSET DELIVERY MECHANISM under ASSETS in Appendix C.

### asset ensemble

A set of assets that will be accessed and used as an aggregate within one customer system. An asset ensemble refines a feature ensemble by determining the architecture of assets and their interconnections that will implement the features of the ensemble. Features of the feature ensemble are mapped to the asset specifications within the asset ensemble. The implementation strategy for each individual asset in the ensemble is defined in the *Implement Asset Base* sub-phase.

#### ASSET IMPLEMENTATION

See ASSET IMPLEMENTATION under ASSETS in Appendix C.

#### ASSET IMPLEMENTATION PLAN

See ASSET IMPLEMENTATION PLAN in Appendix C.

#### ASSET INTERFACE SPECIFICATION

See ASSET INTERFACE SPECIFICATION under ASSETS in Appendix C.

#### ASSETS

See ASSETS in Appendix C.

#### ASSET SPECIFICATIONS

See ASSET SPECIFICATIONS in Appendix C.

#### ASSET SUPPORTING MATERIAL

See ASSET SUPPORTING MATERIAL under ASSETS in Appendix C.

#### ASSET TEST AND VALIDATION PLAN

See ASSET TEST AND VALIDATION PLAN under ASSETS in Appendix C.

### borderline exemplar

A system that has features that provide evidence both for membership and non-membership in the domain, but not sufficient evident for either.

### candidate customer context

A customer that satisfied initial criteria to be considered for selection in the *Scope Asset Base* sub-phase. The decision is made in the final task, *Select Features and Customers*.

#### CANDIDATE CUSTOMER INFORMATION

Supplemental information from potential customers about feature preferences and priorities. CANDIDATE CUSTOMER INFORMATION includes configurations of features that must be available as an ensemble and features that must be excluded from ensembles for them to be usable.

### candidate domain <of focus>

A domain of interest that satisfies initial criteria to be considered as a domain of focus for a domain engineering project.

### CANDIDATE DOMAINS MATRIX

See CANDIDATE DOMAINS MATRIX under DOMAIN SELECTION in Appendix C.

### CANDIDATE EXEMPLAR SYSTEMS

See CANDIDATE EXEMPLAR SYSTEMS in Appendix C.

### CANDIDATE FEATURE — CUSTOMER MAP

See CANDIDATE FEATURE — CUSTOMER MAP in Appendix C.

### candidate objective

A candidate for consideration as a domain engineering project objective; identified in the *Identify Candidate Project Objectives* task and selected in *Select Project Stakeholders and Objectives.*

### CANDIDATE PROJECT OBJECTIVES

See CANDIDATE PROJECT OBJECTIVES in Appendix C.

### CANDIDATE PROJECT STAKEHOLDER MODEL

See CANDIDATE PROJECT STAKEHOLDER MODEL in Appendix C.

### candidate project stakeholders

Organization stakeholders considered for selection as project stakeholders. Determined in the *Select Candidate Project Stakeholders* task; final selection is made in the *Select Project Stakeholders And Objectives* task.

### CANDIDATE REPRESENTATIVE SYSTEMS

See CANDIDATE REPRESENTATIVE SYSTEMS under REPRESENTATIVE SYSTEMS SELECTION in Appendix C.

### CANDIDATE STAKEHOLDER KNOWLEDGE

Used to elicit terms used within the stakeholder community to discover candidate project objectives and domains of interest. Includes:

- literature which discusses or classifies systems of interest to candidate stakeholders, e.g., survey articles or tutorials by acknowledged experts; or
- direct interviews with informants that probe what domains are considered significant within the organization context.

For example, discussions with stakeholders may elicit the term "fault-tolerant mission-critical embedded systems." The objective would then be to identify the specific set of systems of interest to the organization within this class; i.e., the real organization domain in question might be "fault-tolerant mission-critical embedded systems maintained by Division X, but not the ones that run on the old XJK-77 systems..." etc.

### canonic term

A lexicon term within a term cluster selected as the standard to be used in normalized references in other lexicon term definitions, and also in the appropriate DOMAIN CONCEPT MODEL. For example, in the Outlining domain, the terms "sub-head", "sub-heading", "child", and "daughter" might be used in different systems to denote the same entity in outline documents. "Sub-head" would be selected as the canonic term, and would be used consistently in all lexicon references. See also term, term cluster

**closure**

A quality attribute of a domain model with respect to a certain set of operations or transformations. Used by analogy to the mathematical notion of closure under a set of operations. Here, the operations are innovative transforms on feature models.

**commonality**

Similar features observed across exemplar systems. See also variability.

**community of practice**

a group of practitioners in a given organization setting that share terminology, knowledge, a set of behaviors and interests in a certain domain. (Borrowed from ethnographic techniques.)

**component**

Denotes a static ASSET within an ASSET BASE, as opposed to a generative ASSET that produces an asset instance through execution of some program. Often used in the context of component-based vs. generator-based implementation strategies.

COMPONENT CONSTRAINTS MAP

See COMPONENT CONSTRAINTS MAP under INTERNAL ARCHITECTURE CONSTRAINTS in Appendix C.

**component sub-domain**

A domain related to the domain of focus by implementing a well-defined portion of the functionality in the domain. Typical sub-domains would correspond to subsystems within the larger system that implements a given domain. For example, check processing might be a sub-domain of the accounting domain.

More formally, a component subdomain implements a *subset* of the features of the domain of focus. It may also address a broader set of application contexts than the domain of focus. In the example above, check processing may also be a sub-domain of many other domains, such as home financial accounting software. See also specialization domain.

**composite workproduct**

A workproduct composed of a set of workproducts that are outputs of more than one task. Used in the context of the ODM process model.

**concept-driven feature extraction**

Eliciting features by linking relevant entities in the DOMAIN CONCEPT MODELS. In this way the feature serves as an initial integration mechanism for the DOMAIN CONCEPT MODELS. See also informal feature-driven feature extraction.

**concept model**

See DOMAIN CONCEPT MODELS in Appendix C.

### Conceptual Relations

See Domain Interconnection Model in Appendix C.

### condition

A suggested type of model to develop of the Domain Concept Models.

### context

Factors that make one instantiation of a process distinct from another. Also refers to information relevant to understanding why a given process was enacted in a particular way or why a particular workproduct or artifact was developed in a particular way, when this information is not incorporated into the workproduct itself. A context has:

- a name
- a [domain context] type
- an associated set of:
  - Tools and Artifacts
  - Processes
  - Practitioners
  - Knowledge in practice (contextual knowledge)

See also system context, domain context.

### Context Map

See Context Map under Domain Dossier in Appendix C.

### contextualize

The activity of eliciting contextual knowledge about a system or artifact and documenting this knowledge in a way that decreases the amount of tacit contextual knowledge will be needed subsequently to interpret the artifact. Documenting code is a classic way to contextualize it.

### contextual knowledge

Knowledge about the history or rationale behind a system or artifact that is not codified in the artifact itself. Artifacts can be distinguished as low-context and high-context data. High-context data requires that the person interpreting the data understands a lot of the implicit context. Low-context data assumes less knowledge on the part of the person interpreting the data. Contextual knowledge is what has to be gathered from practitioners to turn high-context artifacts into low-context artifacts.

### contextual profile

A set of attributes describing potential application contexts where a particular set of features might be useful. Generated in the *Extend Domain Model* task, in association with specific feature sets. See also potential market profile.

### core customer

A stakeholder that must be satisfied as an asset base customer in order for the domain engineering project to be successful. Usually mandated in the PROJECT CHARTER or otherwise imposed by the

PROJECT CONSTRAINTS, PROJECT OBJECTIVES; or, most informally, the ORGANIZATION CONTEXT. Identifying core customers is one strategy for performing the tasks in the *Scope Asset Base* sub-phase. (Also occurs as core asset base customer.)

**core exemplar**

A system that modelers believe strongly defines or characterizes the essence of the domain (e.g., an established industry lead product).

**core feature**

A feature of interest to a core customer. Alternatively, one of the necessary inclusive DOMAIN DEFINING FEATURES. See core customer.

**core ODM process model**

The core ODM process model addresses processes and workproducts that specifically address domain engineering concerns. The core is represented by a process model that can be tailored and instantiated in a variety of sequences and project structures. Since each organization and each domain will lead to unique constraints and preferences, the core domain engineering process model can be integrated with a broad variety of supporting methods.

**counter-exemplar**

A candidate exemplar system with features identified that provide sufficient evidence of non-membership in the domain.

**customer**

See asset base customer. (Informal usage)

**CUSTOMER APPLICATION SCHEDULES**

Information about the schedules of customer application projects, used to help determine what types of assets might be marketed to them and what schedule constraints would be imposed by their schedule. For example, a given project might be willing to be users of reusable design templates, but only if the templates were available in advance of their completion of requirements analysis. Input to the *Plan Asset Base Implementation* task.

**customer context**

One of the system contexts of a customer application project in which assets might be utilized. For example, the system in development context of a company that would utilize components in building applications. The system in use context could also be considered one of the customer contexts.

**CUSTOMER CONTEXT DOSSIER**

See CUSTOMER CONTEXT DOSSIER in Appendix C.

**customer-driven**

A strategy for selecting the set of features to be supported in the ASSET BASE MODEL, by beginning with core customers and then using these customers to establish a corresponding set of core features to support. See core customer, core feature, feature-driven.

### CUSTOMER — EXTERNAL INTERFACE MAP

See CUSTOMER — EXTERNAL INTERFACE MAP under EXTERNAL ARCHITECTURE CONSTRAINTS in Appendix C.

### DATA ACQUISITION PLAN

See DATA ACQUISITION PLAN in Appendix C.

### deciding interest

Typically, the earliest practitioner in the domain-specific software life cycle who has visibility to a given feature also decides the variant that will be supported. This can be considered the deciding or controlling interest in a given feature.

### defining feature

A statement that establishes criteria to determine the membership of any candidate exemplar system in the domain. Documented in the DOMAIN DEFINING FEATURES MODEL. Later in the process, a defining feature informally denotes a necessary inclusive feature; that is, a feature that holds for every domain exemplar. These domain defining features become the root categories of individual DOMAIN FEATURE MODELS in the *Develop Feature Models* task. See also defining rule.

### DEFINING FEATURES MAP

See CANDIDATE EXEMPLAR SYSTEMS in Appendix C.

### defining rule

A statement about various combinations of features that imply membership in the domain. A defining rule can have the following attributes (where "feature" can also refer to a feature combination):

— Inclusive: the feature holds for exemplars

— Exclusive: the feature does *not* hold for exemplars

— Necessary: the feature holds/does not hold for *all* exemplars

— Sufficient: All systems for which the feature holds/does not hold are exemplars

All of these rules are intensional: that is, they express inherent attributes of the domain, and not merely features that can be observed of a particular set of exemplars but might be accidentally shared attributes, not relevant to the domain definition. See also intensional.

### descriptive feature

A feature value associated with one of the REPRESENTATIVE SYSTEMS SELECTION for the domain. Contrasted with an innovative feature, which may not correspond to a representative system; and a prescriptive feature, which is a feature selected to be supported in the asset base. See also innovative feature, prescriptive feature.

### descriptive feature model

One of the DOMAIN FEATURE MODELS produced in the *Develop Feature Models* task.

### descriptive modeling

In descriptive modeling, knowledge about the domain is obtained in part by "re-contextualizing" legacy systems through the intermediary definition of domain features, significant differentiating capabilities across domain systems. Modelers document commonality and variability in system structure and function and attempt to recapture the rationale for decisions embedded in those systems. They document what practitioners have learned about how to build particular classes of applications.

When used informally rather than to refer to the specific *Develop Descriptive Models* sub-phase, this term usually refers to the descriptive "mind-set" or approach to modeling, as contrasted with the prescriptive engineering approach.

### DESCRIPTIVE MODELS

See DESCRIPTIVE MODELS in Appendix C.

### DESCRIPTIVE MODEL TYPES

See DESCRIPTIVE MODEL TYPES under DOMAIN MODEL ONTOLOGY in Appendix C.

### differentiating feature

A feature that holds for only some exemplar systems. Typically, but not always, a differentiating feature can be modeled as a specialization of a defining feature for the domain.

> *Example.* In the Outlining domain, all outlining programs allow for expansion of sub-heading structure. This would be a defining feature of the domain. However, some outlining programs might include an "expand-all" operation and some might not. This specializes the defining feature; that is, no outlining program could support expand-all that did not support the more general expand operation; while the reverse is not true. Expand-all therefore is a differentiating feature for the domain.

In general, while domain defining features are modeled in the *Define Domain* sub-phase of *Plan Domain Engineering*, differentiating features are not modeled systematically until the *Develop Descriptive Models* sub-phase of *Model Domain.*

### distributed domain

A type of horizontal domain relation. A domain D is distributed with respect to a system S if the functionality of D is pervasive or global within S. A domain D is distributed with respect to a set of systems Sn if the functionality of D is pervasive or global within all systems in Sn. Note that, in ODM, this term is only meaningful as a relation between a domain definition and a system or set of systems. See also horizontal domain, encapsulated domain, vertical domain.

### domain

An abstraction that groups a set of software systems, or some functional areas within systems according to a domain definition shared by a community of stakeholders. The domain can be considered to include not only the shared terminology and definitions, but the coherent body of knowledge about domain systems shared by that community. A domain can be:

- represented by a domain model;
- exemplified by a set of exemplar systems;
- defined by a set of domain defining features and defining rules;
- implemented by an asset base.

See also domain of interest, domain of focus, organization domain

**DOMAIN AFFINITY DIAGRAM**

See DOMAIN AFFINITY DIAGRAM under DOMAIN INTERCONNECTION MODEL in Appendix C.

### domain analysis (non-ODM term)

The process of identifying, collecting, organizing, analyzing, and modeling domain information by studying and characterizing existing systems, underlying theory, domain expertise, emerging technology, and development histories within a domain of interest. A primary goal is to produce domain models to support the development and evolution of domain assets.

NOTE: Since this term is used in ambiguous ways in the domain engineering community the term is avoided in this document. Instead, the ODM domain engineering life cycle includes planning, domain modeling and asset base engineering phases.

### domain architecture/domain architecture model (non-ODM term)

Since these terms are used in ambiguous ways in the domain engineering community the terms are avoided in this document. See instead: exemplar system architecture, asset base architecture, architectural feature model. The closest ODM equivalent to the most common interpretation of the term "domain architecture" would be a "prescriptive architectural feature model."

### domain artifacts

See artifacts; in ODM the term artifact refers to domain artifacts unless stated otherwise.

**DOMAIN ATTRIBUTE DEFINITIONS**

See DOMAIN ATTRIBUTE DEFINITIONS under DOMAINS OF INTEREST in Appendix C.

**DOMAIN CONCEPT MODELS**

See DOMAIN CONCEPT MODELS in Appendix C.

### domain confidante

A key domain informant who can tell you what artifacts are worth looking at, who is considered an expert within the domain, what other informants to talk to, etc. In some cases, the confidante will also be a sponsor who can facilitate the access to the information sources, but these roles can be distinct as well. A "meta-informant." See also informant.

### domain context

A type of system context that occurs in one or more exemplar systems for the domain. Just as we say a system is an exemplar of a domain, a system context can be an exemplar of a domain context. For example, in the domain of Outlining programs, one domain context might be the developer environment in which an outlining program sub-system would be integrated into a full-scale text-editing application. There may be several exemplar systems that have such a context in their specific system life cycle; but not all necessarily will include this type of context. Usually, we make the set of domain contexts be a superset or union of all the distinct types of system contexts for exemplar systems in the domain. This is documented in the DOMAIN CONTEXTS MODEL, part of the DOMAIN INTERCONNECTION MODEL produced in the *Define Domain* task.

### DOMAIN CONTEXTS MAP

See DOMAIN CONTEXTS MAP under DATA ACQUISITION PLAN in Appendix C.

### domain defining feature

See defining feature. In ODM, the term defining feature means domain defining feature unless stated otherwise.

### DOMAIN DEFINING FEATURES

See DOMAIN DEFINING FEATURES under INTENSIONAL DOMAIN DEFINITION in Appendix C.

### DOMAIN DEFINITION

See DOMAIN DEFINITION in Appendix C.

### DOMAIN DEFINITION LEXICON

See DOMAIN DEFINITION LEXICON under INTENSIONAL DOMAIN DEFINITION in Appendix C.

### DOMAIN DEFINITION STATEMENT

See DOMAIN DEFINITION STATEMENT under INTENSIONAL DOMAIN DEFINITION in Appendix C.

### DOMAIN DOSSIER

See DOMAIN DOSSIER in Appendix C.

### domain engineer

An engineer that participates in one or more of the phases of the domain engineering life cycle. In the *Plan Domain Engineering* phase domain engineers may take on the role of domain [engineering project] planner (as distinct from typical project planning); in the *Model Domain* phase the engineer takes on the role of domain modeler; in the *Engineer Asset Base* phase the domain engineer takes on the role of asset base engineer. More specific roles apply within each phase See also domain planner, domain modeler, asset base engineer.

### domain engineering

The creation of new technological and human systems for managing domains within some ORGANIZATION CONTEXT, potentially transforming both the organizations and technical systems within that context. Domain engineering includes creating a DOMAIN MODEL, ASSET BASE ARCHITECTURE and reusable ASSETS for a particular domain. ODM incorporates a life cycle and process model for domain engineering which is described by the ODM Process Model explained in Part II of this document.

### domain engineering life cycle

The domain engineering life cycle contains processes and workproducts that specifically address domain engineering concerns. These domain engineering processes together form the core ODM process model. This life cycle is distinct from and orthogonal to the system development life cycle. Processes that do not address domain engineering are allocated to sets of supporting methods. See also core ODM process model, supporting methods.

### [domain engineering] project

A project that performs an instantiation of the domain engineering life cycle. The project cannot be defined in terms of a project based around a particular domain, since domain selection is part of the life cycle. The project may be part of a larger reuse effort that includes, for example, a system reengineering effort, a corporate reuse planning effort, etc. The project refers in this case specifically to that portion of the larger effort committed to performing the activities within the scope of a domain engineering effort.

### DOMAIN FEATURE MODELS

See DOMAIN FEATURE MODELS in Appendix C.

### domain functionality

An informal term that denotes that portion of functionality within the scope of the domain that occurs within any given exemplar system. Alternative terms that may occur in this document might include "domain system," "domain functional scope," "domain subsystem," "domain instance," "domain instantiation," or "domain-in-system". (This is clearly a term cluster in desperate search for a canonic term!)

### <domain> informant

An individual affiliated with specific stakeholder organizations who is accessed as sources of information during the Domain Modeling phase of domain engineering. Informants' knowledge and experience can be characterized in terms of roles they have played within various contexts for different systems. For example, a veteran designer could be characterized as having played a role in a number of the design efforts performed by a given company.

Informants may be accessed via direct interaction (interviews, group meetings), via targeted surveys and questionnaires, or indirectly through being the source of an article or document used in data acquisition. In the latter case the informant usually brings some perspective other than that of an individual system.

### DOMAIN INFORMANT KNOWLEDGE

Knowledge elicited from domain practitioners in interviews or via observation and process capture. Knowledge can include knowledge about specific representative systems or more general knowledge about domain concepts, terminology, etc.

### DOMAIN INFORMANT MODEL

See DOMAIN INFORMANT MODEL in Appendix C.

### DOMAIN INFORMATION

Information about the domain derived from EXEMPLAR SYSTEM ARTIFACTS or DOMAIN STAKEHOLDER KNOWLEDGE.

### DOMAIN INTERCONNECTION MODEL

See DOMAIN INTERCONNECTION MODEL in Appendix C.

### DOMAIN LEXICON

See DOMAIN LEXICON in Appendix C.

**DOMAIN MODEL**

See DOMAIN MODEL in Appendix C.

**[domain] modeler**

A person performing modeling activities in the context of an ODM project. Strictly, a domain engineer performing activities in the *Model Domain* phase. In usage, sometimes used as a synonym for domain engineer.

**domain modeling**

Development of the DOMAIN MODEL. Also used to denote the general cognitive skill of descriptive domain modeling and taxonomic modeling as differentiated from system modeling.

**DOMAIN MODEL INTERCONNECTION MAP**

See DOMAIN MODEL INTERCONNECTION MAP under DOMAIN MODEL ONTOLOGY in Appendix C.

**DOMAIN MODEL LEXICON**

See DOMAIN MODEL LEXICON in Appendix C.

**DOMAIN MODEL ONTOLOGY**

See DOMAIN MODEL ONTOLOGY in Appendix C.

**DOMAIN MODEL RATIONALE**

See DOMAIN MODEL RATIONALE in Appendix C.

**domain [of focus]**

The domain selected for performing domain engineering.

**domain of interest**

A domain of interest to stakeholders within a given organizational context.

**domain relation/interconnection**

A relationship between the domain of focus and a related domain. See analogy domain, specialization sub-domain, component sub-domain, application context.

**DOMAIN SELECTION**

See DOMAIN SELECTION in Appendix C.

**DOMAIN SELECTION CRITERIA**

See DOMAIN SELECTION CRITERIA in Appendix C.

**DOMAINS OF INTEREST**

See DOMAINS OF INTEREST in Appendix C.

**DOMAINS OF INTEREST CHARACTERIZATION**

See DOMAINS OF INTEREST CHARACTERIZATION under DOMAINS OF INTEREST in Appendix C.

**DOMAINS OF INTEREST LEXICON**

See DOMAINS OF INTEREST LEXICON under DOMAINS OF INTEREST in Appendix C.

### domain-specific

Semantics particular to the domain of focus; i.e., not specific to an individual system, nor a general part of the representations used. The distinctions are in reality more of a spectrum than a hard and fast line. For example, a model of the Ada data type hierarchy is not specific to a particular application area, but is specific to Ada-based systems.

### domain-specific architecture model

(non-core term) The closest equivalent in ODM terms to the conventional notion of a "generic architecture"; a model in an architectural representation language that is claimed to be general enough to serve as the architecture for many systems to be developed for that domain.

### domain-specific meta-model

Synonym for the set of DOMAIN MODELS formalized in the DOMAIN MODEL ONTOLOGY, which forms a "model of models" for the domain. It specifies the scope for each potential model, and relationships and interconnections between the models. Modelers determine which DOMAIN MODELS (of many related models) to build, the level of detail at which to elaborate the models, and any necessary coordination between the models

**DOMAIN SPECIFIC PROJECT OBJECTIVES**

See DOMAIN SPECIFIC PROJECT OBJECTIVES under PROJECT OBJECTIVES in Appendix C.

### domain-specific system life cycle model

The superset of system context types that occur within systems in the domain, and their range of possible relationships. In particular, the interesting details of what happens to software between the developer and the end-user that are peculiar to each domain. Represented by the DOMAIN CONTEXTS MAP.

### domain stakeholder

A stakeholder with interests in the domain models and assets to be produced by the domain engineering project

Some project stakeholders may not be domain stakeholders. For example, a reuse technology developer will generally not be a stakeholder in the selected domain.

Once selected, the domain will also define certain stakeholders who may not have been noted as explicit project stakeholders before.

Thus the transitions from organization to project, and from project to domain stakeholders, are analogous to the transition between descriptive and prescriptive modeling. Primarily a scoping and subsetting operation is involved, with, however, the possibility of some new additions as well.

### DOMAIN STAKEHOLDER KNOWLEDGE

Inventory of artifacts that are not part of exemplar systems and informants available for the domain. These include secondary data sources (survey articles, textbooks, etc.) that may not be associated with particular exemplar systems.

### DOMAIN STAKEHOLDER MODEL

See DOMAIN STAKEHOLDER MODEL in Appendix C.

### DOMAIN TO SYSTEM MAP

See DOMAIN TO SYSTEM MAP under DOMAINS OF INTEREST in Appendix C.

### encapsulated domain

A horizontal domain where functionality is clustered in one structural component of the system in question. See horizontal domain.

### environmental characteristic

A "feature" or distinguishing characteristic about the assumed real-world or outer system context in which a system in the domain will be operating. Could include information like constraints on timing response (e.g., in the telephony domain), noise attributes of lines, etc. Generally outside the direct control of developers.

### ethnographic technique

A technique for gathering information from people via face-to-face interview, observation, participant observation or other relatively non-invasive methods that preserve some of the contextual information otherwise easily passed over. Described in detail in the Information Acquisition Techniques method in Section 8.2.

### exemplar

See exemplar system. An informal term that can be ambiguous, since exemplar artifacts are also frequently referred to. Exemplar system is the preferred term.

### exemplar context

See exemplar system context. An informal term that's reasonably unambiguous.

### exemplar system

A system or subsystem in the scope of which domain functionality occurs. Exemplar systems are enumerated in the EXEMPLAR SYSTEM SELECTION.

NOTE: The term "exemplar system" is something of a misnomer. In the case of a domain that is vertical with respect to a given exemplar system, the entire system is considered within the domain scope; the domain is scoped in terms of sets and subsets of systems. For horizontal domains, or domains that include slices of functionality within larger applications, use of the term "exemplar system" is ambiguous: it can refer to only the domain functionality where it intersects the system, or it can refer to the system as a whole in some cases. Since some domain information gathering activities are planned at the level of overall systems (e.g., identifying experts and other informants for the system, obtaining documentation) this proves a useful term. When the intent is simply to denote the system, subsystem or embedded functionality pertaining to the domain, the less restrictive term "exemplar" by itself is preferable.

See also counter-exemplar, borderline exemplar, core exemplar

### exemplar system architecture

See exemplar system architecture artifacts.

### exemplar system architecture artifacts

Exemplar system artifacts that happen to be architectural diagrams or models.

### EXEMPLAR SYSTEM ARTIFACTS

Artifacts drawn from any phase of the life cycle, from an exemplar system for the domain.

### exemplar system contexts

Let system S be an exemplar of domain D with set of system contexts S1 through Sn. Only certain system contexts will be contexts of interest with respect to D; namely, those that correspond to a context type in the DOMAIN CONTEXTS MODEL. These are exemplar system contexts. That is, the system is an exemplar system with respect to the domain, and the system context is one relevant to the domain.

### exemplar system life cycle

The set of contexts in which a system is developed and used. This set can be represented by a system contexts map, which shows all relevant contexts for the system and their interconnections. See also exemplar system contexts, system contexts map.

### EXEMPLAR SYSTEMS SELECTION

See EXEMPLAR SYSTEMS SELECTION in Appendix C.

### EXTENDED DOMAIN MODEL

See EXTENDED DOMAIN MODEL in Appendix C.

### EXTENDED FEATURE MODEL

See EXTENDED FEATURE MODEL under EXTENDED DOMAIN MODEL in Appendix C.

### EXTENDED STAKEHOLDER MODEL

See EXTENDED STAKEHOLDER MODEL under EXTENDED DOMAIN MODEL in Appendix C.

### extensional

A term borrowed (with apologies) from the field of logic. Defining a category by enumerating examples to suggest what membership means. Used in conjunction with intensional methods of defining rules. See also extensional domain definition.

### extensional domain definition

The extensional domain definition is contained in the EXEMPLAR SYSTEMS SELECTION.

**external architecture**

The set of interfaces that allow a system, sub-system or component to perform its role within a given application context. For a sub-system, it is the shape of the "slot" into which the sub-system fits within the outer system.

EXTERNAL ARCHITECTURE CONSTRAINTS

See EXTERNAL ARCHITECTURE CONSTRAINTS in Appendix C.

**feature**

A statement, usually in semi-formal text, about a capability or attribute associated with an exemplar system workproduct. A feature is used to express a differentiating behavior or characteristic associated with a system artifact that is significant to some domain practitioner within a given context. Each feature is true or not true for each exemplar system. See defining feature, differentiating feature.

**feature binding site**

Points in the workflow at which feature variants may be selected. Depending on the domain context and the feature binding site, this selection may be done programmatically, by software, or by human decision-making. Different representative systems will make different choices with regard to where feature variants are bound. Some systems will have sites that do not occur elsewhere. Some will provide multiple options.

FEATURE BINDING SITES MAP

See FEATURE BINDING SITES MAP under INTEGRATED DOMAIN MODEL in Appendix C.

**feature cluster**

Synonym for feature set, or feature combination. Informally, used in the *Integrate Descriptive Models* task to denote specific collections of feature variants that specialize a given feature category. Thus clusters are smaller than individual FEATURE MODELS, but are aggregates rather than individual features.

**feature combination**

Synonym for feature set, or feature ensemble. Ambiguous. Used in the *Extend Domain Model* task to denote feature sets that are interim constructs generated from applications of transformation rules; feature set is used later in the life cycle (e.g., the *Correlate Features and Customers* task) to denote sets that have been judged as coherent and stable as a whole.

**feature configuration**

Synonym for feature set, or feature ensemble. Ambiguous.

**feature distance**

A metric for how much semantic distance exists between two feature variants in a model.

**feature-driven**

In the *Correlate Features and Customers* task: Beginning with identified sets of features and using these features to determine associated contexts and specific customers that match the profile of those contexts. Contrasted here with customer-driven. See core customer, core feature, customer-driven.

Also: selecting prescriptive features using a bottom-up feature-driven approach of describing feature variants to be supported for individual asset-level functionality. See architecture-driven.

### feature-driven feature extraction

In the *Develop Feature Models* task: Each DEFINING RULE is converted into canonic lexical form using terms from the DOMAIN MODEL LEXICON and then converted directly to a feature. The feature is then analyzed into its constituent concepts. This yields a sparser DOMAIN CONCEPTS MODEL that contains only concepts necessary for defining features. See also concept-driven feature extraction.

### feature ensemble

A feature set that can be realized as a single implementation made available within a system context. This could be the feature profile for an entire system instance within the domain (i.e., an architectural variant) or an ensemble to be implemented by multiple components made available as a library (e.g., a set of system services) within a system-in-development context.

### FEATURE ENSEMBLE CONSTRAINTS

See FEATURE ENSEMBLE CONSTRAINTS under INTERNAL ARCHITECTURE CONSTRAINTS in Appendix C.

### FEATURE ENSEMBLE — CUSTOMER CONTEXT MAP

See FEATURE ENSEMBLE — CUSTOMER CONTEXT MAP under INTERNAL ARCHITECTURE CONSTRAINTS in Appendix C.

### FEATURE — EXTERNAL INTERFACE MAP

See FEATURE — EXTERNAL INTERFACE MAP under EXTERNAL ARCHITECTURE CONSTRAINTS in Appendix C.

### feature profile

In general, a set of feature statements that hold for a particular artifact or asset. A descriptive feature profile is a feature set that characterizes a particular representative system. Descriptive feature profiles are created after the feature sets are combined into the INTEGRATED DOMAIN MODEL.

A prescriptive feature profile is a set of features allocated to a specific asset specification, part of the ASSET BASE ARCHITECTURE or later ASSET IMPLEMENTATION PLAN.

### feature set

At its most general, a feature set can be any collection of features within a DOMAIN FEATURE MODEL. Feature sets become important in the *Extend Domain Model* task, where feature sets are identified that satisfy certain formal properties of closure, cohesion and orthogonality. Each of these feature sets is associated with attributes of contexts in which the feature sets could be useful to practitioners; these are the potential market profiles for the feature sets. See also feature profile.

### feature space

Informal term denoting the range of feature combinations possible for a given set of feature variants.

### feature variant

An immediate specialization of a feature within the feature model. An arbitrary descendant is a **feature specialization.**

### flexible architecture

See highly flexible architecture.

### formal feature

A feature statement in which each key domain-specific term or phrase is linked to elements in the DOMAIN CONCEPTS MODELS.

### forward system engineering

See system engineering.

### genealogy

A model that shows the historical relationships between representative systems in the domain. In particular, it shows relations relevant to reuse and domain engineering, e.g., leveraged code, requirements reused on new platforms, etc. The classic information captured by such a model would be a system family relationship. The term genealogy is meant to suggest that more remote "family relationships" can also be important to note.

This information is used in two tasks in particular: in *Plan Data Acquisition*, in order to select a suitably diverse REPRESENTATIVE SYSTEMS SELECTION; and in *Interpret Domain Model*, when the information is used in order to correctly interpret commonality and variability observed across the representatives.

### generalization

An entity generalizes another entity when it has *fewer* features or attributes, and/or addresses a broader context of application. See also specialization.

### generalization domain

A domain D1 is said to *generalize* a domain D2 if:

- The exemplar set, or organization context, of D1 is a superset of the exemplar set of D2;
- The set of defining features for D1 is a *subset* of the defining features for D2 (i.e., a less tightly constrained definition).

### generative asset

See generative technology.

### generative implementation

See generative technology.

### generative technology

Use of programming techniques that allow implementations to be generated from a high-level specification. In the ODM context, the specification is typically a domain-specific specification utilizing terms from the lexicon and/or DOMAIN FEATURE MODELS.

### generator-based implementation strategy

See generative technology.

### group

Some sub-division of an organization that includes a specific group of people: a department, team, division. Groups can have sub-groups to arbitrary levels.

### highly variable architecture

An architecture that supports both variability by replacing or masking components, but via structural variation in the topology of the architecture itself. In particular, highly variable architectures may need to provide both optimized and non-optimized versions of particular component configurations, simultaneous access to multiple structural layers for certain configurations, etc.

### horizontal domain

A domain that includes only a subset of the functionality implemented in a system. Horizontal domains may be encapsulated or distributed. In ODM the terms *vertical* and *horizontal* are not absolute qualities of general functional areas, but describe the span of a given domain's coverage relative to a particular set of systems. For example, within a family of large-scale systems, database management may be considered a horizontal domain. In the Database Management Systems provider marketplace, database management could be considered a vertical domain. See vertical domain, encapsulated domain, distributed domain.

### INFORMAL DOMAIN DEFINITION STATEMENT

See INFORMAL DOMAIN DEFINITION STATEMENT under DOMAIN SELECTION in Appendix C.

### informal feature

A feature statement describing a particular representative system or system artifact, elicited from an informant or by examining an artifact, and not expressed using a constrained vocabulary.

### INFORMAL FEATURE SETS

See INFORMAL FEATURE SETS under DOMAIN DOSSIER in Appendix C.

### informant

A stakeholder (usually a practitioner) selected as a source for domain information.

### INFORMANT DATA

See INFORMANT DATA under DOMAIN DOSSIER in Appendix C.

### INFORMANT ROLES IN CONTEXT

See DOMAIN INFORMANT MODEL in Appendix C.

### information

can be quantitative or qualitative in nature...

### INFORMATION ACCESSIBILITY — QUALITY MAP

See INFORMATION ACCESSIBILITY — QUALITY MAP under DATA ACQUISITION PLAN in Appendix C.

### information-driven modeling

A domain modeling approach in which the modeler begins with an information source, and attempts to filter relevant data from that information source into the appropriate models. One advantage of this approach is that certain information sources can be very thoroughly modeled, and may even suggest new and important model types not originally planned for in the DOMAIN MODEL ONTOLOGY. One disadvantage is the difficulty in regulating the appropriate level of detail to model.

### information source

A specific document or informant used as a source of information for domain engineering. Information sources are instances of one or more information types.

### information source type

A kind or class of information available within the domain, characterized in terms of the medium, the format, the life cycle phase with which it is associated (e.g., requirements document, test plan), or other criteria. The term can include not only documents and system artifacts but people.

### INFRASTRUCTURE IMPLEMENTATION PLAN

See INFRASTRUCTURE IMPLEMENTATION PLAN in Appendix C.

### initiative

Within the overall context of reuse programs, the term refers to: 1) the meta-process by which a reuse program is created within a given organization context; and 2) the first cycle or sequence of cycles through the Reuse Management processes for the program as a whole, during which the program serves the dual purpose of creating, managing and motivating reuse of asset bases, and improving the organization's capability of performing reuse-based software engineering.

The initiative can be seen as a special case of a transition function, relative to a model of capability or maturity levels, where an initiative moves the organization from stable performance at one level to a new level of performance. In this regard, a special significance is given to the first initiative, where the organization moves from the informal reuse state to a managed state. This is not a simple sequence of activities to characterize, since different organizations will start from different levels even in terms of their informal reuse practice.

### innovative feature

A feature that was generated by a direct application of a transformation rule on a feature model, rather than by examination of domain data. The innovative feature need not correspond to any particular representative system.

### instance

A specific individual of a given type. A system context is an instance of a system context type; an exemplar feature is an instance of a feature category or formal feature.

### instantiation

The process of or result of creating an instance. See also instance.

#### INTEGRATED DOMAIN MODEL

See INTEGRATED DOMAIN MODEL in Appendix C.

#### INTEGRATED FEATURE MODEL

See INTEGRATED FEATURE MODEL under INTEGRATED DOMAIN MODEL in Appendix C.

### intensional

An intensional rule expresses inherent attributes of the domain, not merely features that can be observed of a particular set of exemplars but might be accidentally shared attributes, not relevant to the domain definition.

> *Example.* If all the exemplars examined for outlining programs ran on the Macintosh, this would not necessarily make the feature "runs on Macintosh platform" part of the intensional definition for the domain. The test to apply is the following: if an exemplar were examined that matched all other features but the one in question, would it be an exemplar or not? If the answer is yes, the feature is extensional, an observed commonality of the exemplar set but not fundamental to the domain definition. If the answer is no, the feature is an intensional, defining feature.

#### INTENSIONAL DEFINITION TO INTERCONNECTION MODEL MAP

See DOMAIN INTERCONNECTION MODEL in Appendix C.

#### INTENSIONAL DOMAIN DEFINITION

See INTENSIONAL DOMAIN DEFINITION in Appendix C.

### interest

A strategic relation or other motivation of a stakeholder in a system, a project objective or a domain. Interests can be both positive (e.g., perceived opportunities or incentives) or negative (e.g., perceived risks, threats or other disincentives).

### internal architecture

The partitioning of a feature ensemble into a set of asset specifications with defined interconnections. Key issues addressed by the internal architecture are separate selectability of various subsets of the constituent assets, masking of components, and layering strategies.

#### INTERNAL ARCHITECTURE CONSTRAINTS

See INTERNAL ARCHITECTURE CONSTRAINTS in Appendix C.

### interpreted feature

A descriptive feature for which rationale and/or process and contextual information has been obtained in the *Interpret Domain Model* task. Generally the interpreted feature has been correlated with some attributes of the representative system contexts in which it occurs as rationale.

**Interpretive Domain Model**

See INTERPRETIVE DOMAIN MODEL in Appendix C.

**key customer**

See core customer. Synonym.

**KEY EXTERNAL INTERFACES**

See KEY EXTERNAL INTERFACES under EXTERNAL ARCHITECTURE CONSTRAINTS in Appendix C.

**key role**

A key stakeholder, informant, or customer role.

**key stakeholder**

A stakeholder whose buy-in and commitment are deemed essential to the overall success of the domain engineering project.

**layer**

Optional extension to the core ODM process model. Layers have a pervasive impact across the entire domain engineering life cycle. See Section 9.0 of this document for a description of some of the layers that can be added to the core ODM process model.

**LAYERING CONSTRAINTS MAP**

See LAYERING CONSTRAINTS MAP under INTERNAL ARCHITECTURE CONSTRAINTS in Appendix C.

**lexicon**

See DOMAIN LEXICON, DOMAIN MODEL LEXICON.

**life cycle**

The top level, or "root" node, of the ODM process tree (i.e., *Domain Engineering*), representing the overall ODM domain engineering life cycle. See domain engineering.

**market**

See customer context.

**mask**

A transformation where the entity in question is removed from the system of which it is a part, resulting in a variant structure which is then analyzed for usability and feasibility. This technique can be used on feature sets in the *Extend Domain Model* task and later, in the *Determine Internal Architecture Constraints* task in the *Architect Asset Base* sub-phase.

**meta-model**

See ontology. Meta-model usually refers to the taxonomic modeling representation language in which the ontology, the starter models, and the actual concept and feature models are developed.

**model**

Within the ODM context, a model implies one of the domain models as opposed to a system model. These domain models are maps of commonality and variability across systems rather than models of the structure or function of a single system. Models are represented using a Taxonomic Modeling Support Method (See Section 8.3). See DOMAIN MODELS in Appendix C.

**model-based development**

Related to domain engineering; in the ODM context, system engineering performed on the basis of a DOMAIN MODEL but not necessarily involving the development of an ASSET BASE MODEL, ASSET BASE ARCHITECTURE, or ASSETS. In these cases, the DOMAIN MODEL is basically being used as a comprehensive superset of requirements in the domain (although requirements assets *per se* have not been engineered. This definition may differ significantly from use of the term in other research.

**model-driven**

In the *Develop Descriptive Models* sub-phase, this strategy uses the MODEL TYPES LIST in the DOMAIN MODEL ONTOLOGY as a filter on the data collected and elicited. It contrasts with data-driven strategies, where the process is organized in terms of artifacts being completely examined and parsed into the relevant models.

**NEW SYSTEM ARTIFACTS**

See NEW SYSTEM ARTIFACTS in Appendix C.

**object**

A real-world entity represented by data or by an artifact within a system context.

**objective**

A domain engineering project objective. See PROJECT OBJECTIVES.

**object model**

One of the suggested concept model types (non-core).

**ontology**

For an individual model, a definition of what the model is talking about; the scope of entities that will be included in the model. A synonym might be model type or model name, although the meanings are not quite the same. The DOMAIN MODEL ONTOLOGY produced in the *Select Descriptive Model Types* task includes an ontology description for each selected model type, and specifies high-level semantic relations among the models. See also DOMAIN MODEL ONTOLOGY.

**ontology-driven modeling**

A domain modeling approach focused around acquiring data to populate a specific model. In this approach, modelers start with a selected model, then sifts through domain information sources for data relevant to this model. One advantage of this approach is that it is easier to control acquiring necessary and sufficient data for each model. One disadvantage is that the same information source may be analyzed numerous times, by different modelers, and/or in the context of different models.

### organization

A company, institution, informal group, community group, etc. Some group of people who voluntarily self-identify as a larger entity for some purpose (business, cultural, social, etc.). See also stakeholder context.

#### ORGANIZATION CONTEXT

The business strategies, policies and procedures, expertise, technological capabilities, cultural legacies, etc., of the set of organizations involved in a reuse effort.

### organization domain

A shared abstraction typically intersecting, and integrating, organizational groups and structures, areas of expertise, and projects oriented around particular software products or systems. An organization domain usually spans several systems, may include only sub-portions of these systems, and will usually map across functional divisions within or even across organizations as well.

#### ORGANIZATION INFORMATION

One of the primary inputs to domain engineering. General sources of ORGANIZATION INFORMATION may include: organization charts, business plans, marketing literature, annual reports, and informal knowledge of project members and domain experts. Other valuable sources of information are results of reuse-specific and broader organization assessments.

### organization setting

A set of practitioners in an organization performing activities, interacting with each other, using tools.

Settings can include sub-settings. There can also be interactions across settings (e.g., a tool developed in one setting can be used in another setting) drawing useful boundaries around a particular setting within an organization is a fundamental task in analysis. The way we define organization settings influences the way we define system contexts.

When we view a particular system from the standpoint of a particular domain, only certain system settings will be contexts of interest with respect to that domain. These are ***exemplar system contexts***. That is, the system is an exemplar system with respect to the domain, and the system context is one relevant to the domain.

### organization stakeholder

A person with interests in an organization. A stakeholder will be a practitioner in at least one setting for that organization. In some cases, that setting may seem quite remote from the usual way of bounding the organization. For example, stockholders are certainly key stakeholders in an organization; the setting in which they are practitioners is the stock market. They generally won't know much about the business though, so they may not be good candidates to be ***informants***.

In ODM, we're only concerned with organization stakeholders that have an interest in the domain engineering project itself. We're not doing stakeholder analysis on the organization as a whole; in addition, many projects will involve multiple organizations. Hence, ***project stakeholders*** are the first key area of focus in the ODM process, in the *Determine Candidate Project Stakeholders* task (1.1.1). ***Candidate project stakeholders*** are identified out of the broader set of organization stakeholders revealed in ORGANIZATION INFORMATION.

**organon**

A historical term that means "a structured body of knowledge". In the context of this document, a longer-term vision of an asset base, founded on the notion of repositories of codified domain knowledge, coupled with pro-active technologies to support the use and evolution of the knowledge. The organon also includes the community of stakeholders who collaboratively define the domain, build the domain model and engineer and use the assets.

**orient**

To characterize a domain with respect to various axes including:

- historical (e.g., predecessor or anticipated successor technologies),
- operational (e.g., subsystems or applications within which domain functions are embedded),
- conceptual (e.g., broader domains, more specialized domains, analogy domains).

Orienting the domain results in the development of a DOMAIN INTERCONNECTION MODEL.

**parse**

The activity of examining an artifact looking for candidate terms for the lexicon. May also include separating out portions of the artifact relevant to the domain; i.e., selecting which modules are within the domain scope.

**partitioning**

Where this term is used, it implies that sub-divisions are non-overlapping and exhaustive.

**phase**

One of the three major components of the ODM life cycle: *Plan Domain Engineering*, *Model Domain,* or *Engineer Asset Base*. Generally these are referred to as the <Domain> Planning, <Domain> Modeling, and <Asset Base> Engineering phases.

**populate**

To add descriptions of exemplar systems or artifacts as ***instances*** into a model in the appropriate categories. This is distinct from adding a new category or relationship to the model itself.

**potential**

Anticipated at a future point in the ODM process, but not at a point where the decision is currently being made. For example, in the *Plan Domain Engineering* phase potential customers may be identified who will not be selected until the *Engineer Asset Base* phase. Once the selection process is initiated (in *Scope Asset Base* sub-phase) candidate customers are considered. See also candidate.

**potential market profile**

Attributes of an application context identified in the *Extend Domain Model* task and associated with a particular feature set. For example, an outline browser supporting read-only browse mode might be of interest in an environment where collaborative document preparation was being performed. The latter attribute becomes an attribute of the potential market profile.

### practitioner

Practitioners are individual people who are members of organizations and/or groups in organizations, and perform activities in organization settings. Practitioners can include developers, operators, users, managers; domain experts and novices. Some practitioners will be selected as informants, and some will be selected as customers.

### precedent

An instance within an EXEMPLAR SYSTEM ARTIFACT that supports the inclusion of a term or a concept in a lexicon or a DOMAIN MODEL.

### prescriptive modeling

The asset base engineering part of the domain engineering life cycle, where binding decisions and commitments about the scope, architecture and implementation of the ASSET BASE are made. The descriptive feature model is transformed into a prescriptive set of committed features for implementation. These prescriptive features are mapped onto the structure of the ASSET BASE ARCHITECTURE and ASSET SPECIFICATIONS. By preserving traceability from features back to EXEMPLAR SYSTEM ARTIFACTS, asset developers can gain access to potential prototypes on which to base development of new ASSETS. In addition, because ASSETS are described within the ASSET BASE in terms of the domain feature model, asset utilizers can eventually retrieve components based on the same descriptive features.

### PRIORITIZED FEATURE — CUSTOMER MAP

See PRIORITIZED FEATURE — CUSTOMER MAP in Appendix C.

### PRIORITIZED INFORMATION TYPES

See PRIORITIZED INFORMATION TYPES under DATA ACQUISITION PLAN in Appendix C.

### process

A set of steps that are carried out to reach a goal.

### process asset

Encoding, formalization, or automation of a work process in the domain, typically used as a tool, decision aid, etc. rather than a reusable component to be directly incorporated into systems. See also artifact-based asset, stakeholder-based asset.

### process history

Capturing information about the changes over time to specific workproducts, or the actual as opposed to planned performance of processes. An important tool in data acquisition within the domain to discover opportunities for encapsulating processes as assets. Also an optional layer of the ODM method, used when incremental improvement of the domain engineering process itself is a desired objective of the project.

### project

In the ODM context, a domain engineering project.

**PROJECT CHARTER**

Provides bounding criteria for the ORGANIZATION CONTEXT of the project, including whether the initiative is part of a more comprehensive reuse program. The charter may be a formal document or may be implicit, based on the informal contextual knowledge of project members.

Charters can take several forms, including the following:

- A mandate to perform domain engineering within a particular business area, selecting the domain and technology as appropriate;
- A mandate to validate or demonstrate a particular reuse technology;
- A mandate to achieve certain business objectives (time to market, productivity improvement) without specifically being directed to take a domain engineering approach.

**PROJECT CONSTRAINTS**

Restrictions imposed by organization and market conditions beyond the project scope. Constraints may include implicit expectations about project results, perceived risks, and other factors that could compromise the success of technically sound results.

Constraints may include the following:

- Technology constraints
- Constraints on domain selection
- Available resources, as well as constraints on resources that can be tapped. For example, there may be a pragmatic constraint that the domain engineering effort cannot have any impact on the schedule or resources available for an application engineering project of high priority. Such a constraint can significantly shape viable objectives for the project.
- Restrictions on the scope of organization boundaries that the project can cross. For example, a domain engineering project in a software maintenance organization may have restricted access to organizations that developed application systems in the domain.

PROJECT CONSTRAINTS are iteratively identified and documented during Planning, as CANDIDATE PROJECT OBJECTIVES raise issues that reveal previously unconsidered or implicit constraints. PROJECT CONSTRAINTS include PROJECT RESOURCES.

**PROJECT — CUSTOMER TIME LINE**

See PROJECT — CUSTOMER TIME LINE under ASSET IMPLEMENTATION PLAN in Appendix C.

**project member**

A domain engineer, a member of the project team.

**PROJECT OBJECTIVES**

See PROJECT OBJECTIVES in Appendix C.

**PROJECT — REPRESENTATIVE TIME LINE**

See PROJECT — REPRESENTATIVE TIME LINE under REPRESENTATIVE SYSTEMS SELECTION in Appendix C.

### PROJECT RESOURCE

Staff, budget, technology, or any other resource to be allocated to domain engineering. A component data flow of PROJECT CONSTRAINT, indicating the general pessimism of project managers.

### project stakeholder

A stakeholder with interests in the intended results of the project as defined by the PROJECT OBJECTIVES.

### PROJECT STAKEHOLDER MODEL

See PROJECT STAKEHOLDER MODEL in Appendix C.

### PROJECT STAKEHOLDER ROLES

See CANDIDATE PROJECT STAKEHOLDER MODEL in Appendix C.

### project team

The team performing the domain engineering project.

### quality criteria

An attribute for evaluating information sources during the PLAN DATA ACQUISITION sub-phase.

### re-contextualizing

See contextualizing.

### related domain

A domain related to the *domain of focus* according to one of the domain interconnection or domain relation types that form part of the DOMAIN INTERCONNECTION MODEL.

### RELATED DOMAINS — DEFINING FEATURES MAP

See RELATED DOMAINS — DEFINING FEATURES MAP under DOMAIN INTERCONNECTION MODEL in Appendix C.

### relation

One of the suggested conceptual model categories (non-core). A second-order model entity based on objects and operations; i.e., object-to-object, object-to-operation, and operation-to-operation relations. For example, in the Outlining Domain, the term "headings" denotes a basic object in the outline structure. The terms "sister" or "sibling" are synonymous terms denoting a particular relation between headings.

### representative exemplar system

See representative system.

### REPRESENTATIVE SELECTION CRITERIA

See REPRESENTATIVE SELECTION CRITERIA under REPRESENTATIVE SYSTEMS SELECTION in Appendix C.

**representative system**

An exemplar system which has been selected for detailed analysis as a primary source of information during the *Develop Descriptive Models* sub-phase.

**representative system artifacts**

Artifacts selected from representative system contexts as sources of information for domain modeling.

**REPRESENTATIVE SYSTEMS FEATURE PROFILE**

See REPRESENTATIVE SYSTEMS FEATURE PROFILE under INTEGRATED DOMAIN MODEL in Appendix C.

**REPRESENTATIVE SYSTEMS SELECTION**

See REPRESENTATIVE SYSTEMS SELECTION in Appendix C.

**REUSE CRITERIA**

General domain-independent criteria for selecting promising domains for reuse.

**reuse management**

The management aspects of reuse addressing the establishment and continual improvement of reuse-oriented activities within an organization by emphasizing learning as an institutional mechanism for change. Learning in this context means actively evaluating and reflecting on behavior to effect positive change.

**reuse program**

The set of activities encompassed by (and including) a particular instance of reuse management.

**reverse engineering**

The process of analyzing a computer system's software to identify components and their interrelationships.

**role**

A position of responsibility to be occupied when performing work. Agents play roles in contexts. See agent, context.

**root category name**

A name associated with a DOMAIN CONCEPT MODEL or DOMAIN FEATURE MODEL which describes the type of entity or concept that will be within the scope of the model. For example, "error message" could be the root category name for a model focused on classifying different types of error messages provided in a system. Alternatively, modelers could choose to embed this name within a more general taxonomy, such as "system condition". The choice of how to partition the set of models is part of the modeling task and not specified by ODM.

**scope of applicability**

The range of application contexts for which an ASSET has been designed. Sometimes termed the intended scope of applicability, application scope, scope of application, market profile.

SELECTION MATRIX

See SELECTION MATRIX under REPRESENTATIVE SYSTEMS SELECTION in Appendix C.

**separate selectability**

The ability of a particular functional subset or subsystem of an overall system to be extracted as a stand-alone system or component. An architecture which allows for various subsystems to be extracted in this way is a **separately selectable architecture.**

**specialization**

A specialization of a given entity has *additional* features and usually addresses a *narrower* scope of applicability. See also specialization subdomain.

**specialization subdomain**

A domain D1 is said to *specialize* a domain D2 if:

- The exemplar set, or organization context, of D1 is a subset of the exemplar set of D2; or
- The set of defining features for D1 is a *superset* of the defining features for D2 (i.e., a more tightly constrained definition).

This is an easy point of confusion. Consider the generality vs. optimization trade-off: the narrower the organization context for which a system is built, the more functionality can be included in the system, e.g., it becomes more special-purpose. So as the context narrows, the set of features expands.

**sponsor**

A stakeholder who funds or otherwise initiates a domain engineering project, and typically writes or controls the PROJECT CHARTER.

**stakeholder**

A person, group or organizational entity that has interests and objectives relative to a system, project, or domain. Stakeholders have diverse viewpoints, experience, and terminology. Stakeholders include organization stakeholders, project stakeholders, and domain stakeholders. In a domain engineering project, these stakeholders can hold the role of a domain engineer, informant, customer, or practitioner.

**stakeholder-based asset**

Closely related to process assets, stakeholder-based assets are structured around the roles, profiles, or experience of particular stakeholders. Since a given stakeholder may utilize multiple workproducts and perform many processes, opportunities for encapsulating reusable knowledge into assets may emerge independently of artifact or process structure. For example, automated or codified expert knowledge could be used in decision making or workproduct transformation.

**stakeholder community**

See stakeholder context.

### stakeholder context

The set of stakeholders that form the ORGANIZATION CONTEXT the domain engineering project, together with their interests and relations. Depending on the nature of the domain engineering initiative and funding for the project, this context can be:

- a single organization or a division of a larger organization, such as a corporate or university research and development facility, a product-line division, or a system development group,
- multiple organizations, as typified by standards organizations and consortia with common interests in particular functional areas,
- a marketplace of varied competitor, partner and customer organizations, or
- a technical community.

Synonymous with organization context. (This context may informally be referred to as "the organization" for convenience.)

### stakeholder interest

A perceived benefit, risk, or other strategic motivator to a stakeholder with respect to the domain engineering project.

### starter model

A model reused from non domain-specific infrastructure, previous domain engineering efforts that provides a starting point for descriptive modeling.

### STAKEHOLDER OBJECTIVES MAP

See STAKEHOLDER OBJECTIVES MAP under PROJECT OBJECTIVES in Appendix C.

### state

Within the suggested concept modeling approach, a state is an aggregate of object structures, process history, and conditions sufficient to characterize the overall domain functionality within a system context. See also condition.

### sub-domain

See component sub-domain and specialization sub-domain. The term "sub-domain" by itself is ambiguous in the ODM context and its use is generally avoided. When it occurs it is informal for component sub-domain or means both types of sub-domain.

### sub-phase

Each of the three phases of domain engineering are further decomposed into sub-phases.

### subsystem

A component of a system; usually but not necessarily encapsulated, i.e., appearing as a distinct component in the system architecture.

### supporting methods

Methods that do not address domain engineering concerns but are needed to carry out domain engineering. These methods should be chosen by each organization based on their constraints, preferences, and domain. Supporting methods include system modeling techniques and taxonomic modeling techniques. A description of supporting methods that must be chosen for domain engineering appears in Section 8.0.

**system**

An application designed for a single context of use. There is no intent to imply a systems (i.e., hardware, software, practitioners) vs. software-only distinction. A system is developed and used by a set of practitioners performing activities, using tools and technology, within a set of system contexts.

**system architecture**

The high level design of a software system. An architecture is defined in terms of the following general constructs:

- a set of software system elements, which may include both processing and data elements
- interfaces for each element
- a set of element-to-element connections, collectively forming interconnection topologies
- the semantics of each connection
  - — the meaning of static connections (e.g., between data elements)
  - — protocols describing information transfer across dynamic connections, in terms of element interfaces (general classes of protocols include procedure call, pipe, message passing, etc.

**system architecture artifact**

An artifact representing the design of a representative system. This may have been a primary artifact, created by the system developers, or a post hoc architectural reverse engineering of a legacy system using an architecture design notation. The latter may be done by the domain engineers in order to have a common notation for comparing representative systems at the architectural level.

**system architecture asset**

An asset designed to be used as an architectural design in developing a system.

**system artifact**

An artifact used in development or execution of a system; e.g., a requirements document, design document, executable module, system start-up file.

**system context**

Each system has a set of system contexts, which link the system and a specific organization context. System contexts include:

- System-in-development context
- System-in-operation context
- System-in-use context

These contexts correspond to settings in which domain activity is performed. See system-in-development, system-in-operation, system-in-use.

**SYSTEM CONTEXTS MAP**

See SYSTEM CONTEXTS MAP under DOMAIN DOSSIER in Appendix C.

**system context type**

A category that groups different contexts that play a similar functional role in the system life cycle: e.g., system-in-development is a context type. There may be several instances of a system context type (called simply contexts, *not* "context type instances") in a given system's life cycle. The set of system context types applicable for a given domain are specified at a high level in the DOMAIN INTERCONNECTION MODEL produced during *Define Domain*. After a REPRESENTATIVE SYSTEMS SELECTION is made, the set of context *types* that will examined to produce the DESCRIPTIVE MODELS is recorded in the SYSTEM CONTEXTS MAP.

**system engineering**

Implementing a new system for a single application context. What constitutes a single application context will vary from organization to organization. Contrasts with variability engineering, which involves a specified set of multiple application contexts, and domain engineering, which involves strategic selection of a set of multiple contexts.

**system-in-development**

The context in which software systems are developed. Typically the programming platform and environment.

**system-in-operation**

That aspect of the system-in-use that involves only interactions of the domain system (i.e., the domain functions within the system) with other system functions, e.g., system interfaces. For a system executing in an unmanned or autonomous settings, the system-in-operation may appear to be virtually the same as the system-in-use.

**system-in-practice**

Can refer to any system context; refers to the layer of human practices and behaviors with respect to the system. In the ODM context, the focus is always on the domain functionality within the system context. For an end-user, the system-in-practice context might include artifacts like a checklist of system commands taped to the terminal. For a system developer, the system-in-practice context might include the log of bug fixes made to a program before resubmitting it to configuration management. See also system-in-development, system-in-use.

**system-in-use**

A context where a software system executes its functions, usually in interaction with human beings. Includes the system-in-operation. See also system-in-operation.

**system practitioner**

A person who plays one or more practitioner roles within one or more contexts of a system. See also practitioner.

**system reengineering**

Restructuring and/or reimplementing some or all of an existing legacy system.

**SYSTEMS OF INTEREST**

An inventory of applications (software, systems, organizational processes) within the project scope. In a development environment, SYSTEMS OF INTEREST could include both legacy systems and anticipated new systems. In a maintenance environment, SYSTEMS OF INTEREST could include the inventory of systems under control of the group initiating the domain engineering project.

### task

One of the 27 "leaf nodes" of the process tree that represent a primary task in the ODM life cycle.

### taxonomic modeling

One of the supporting methods described in Section 8.0. Taxonomic modeling is used to describe the commonality and variability in a set of exemplars. It may involve hierarchical classification schemes, more formal semantic networks, simpler faceted schemes, or other representations. It contrasts, in the ODM context, with system modeling, which focuses on structural and behavioral descriptions of individual systems. In practice some integration between the two modeling representations must be supported. Variability modeling is a specialization of taxonomic modeling used in the ODM context to specifically addresses variability within software systems. See also variability modeling, system modeling.

**TECHNOLOGY CONSTRAINTS**

A component flow of PROJECT CONSTRAINTS; constraints that affect the selection of a supporting method in one the areas identified in Section 8.0.

### team modeling

One of the optional layers described for ODM.

### term

An entry in the DOMAIN LEXICON or DOMAIN MODEL LEXICON.

### term cluster

A group of terms interpreted by the modeler as synonymous, and gathered together in a term cluster, with a a standard canonic term as a representative. See also canonic term.

The term cluster may include acronyms or contextual abbreviations of compound terms as well as true synonyms. For example, "domain exemplar" and "exemplar" could be considered a cluster, where "domain exemplar" more fully qualifies the term "exemplar". Within a given domain context, a given term may be assigned a semantics that overlaps with the most commonly intended compound term; so, for example, it is not necessary in the ODM context to mention domain exemplars; but use of exemplar without the meaning of "domain exemplar" would need to be made explicit. Other syntactic variations can be encompassed as part of the term cluster: e.g., case, voice (e.g., active-passive) and plurality variants, etc. (For example: "variable" and "variability," "bounds" and "bounded by," "concept model" and "concept models.")

**TEST AND VALIDATION PLAN**

See TEST AND VALIDATION PLAN in Appendix C.

### usability

In the context of ODM, evaluation of whether a given feature or feature combination would serve a given function within a given system context. For example, in the Outlining domain, an outline program that provided expansion but not collapsing of sub-headings could be assessed for usability relative to a given group of end-users.

### validation

In the ODM context: testing a model against empirical data. An extended form of validation tests a model developed to a given set of data for adequacy against appropriate data not used in its development. If the new data does not fit the model, the model must be adjusted or the data rejected as either in error or beyond the intended scope of the model. See also adequate.

### variability

The ways in which two concepts or entities differ. Used in opposition with commonality. In the ODM context, See also commonality.

### variability engineering

Application and extension of systems engineering techniques to multiple-system contexts, such as product-line planning and engineering. Encompasses variability modeling, as well as various design and implementation techniques to cost-effectively support variability, via flexible architectures, generative technologies, etc. Variability engineering is a sub-problem of domain engineering; the term implies that there is an established set of customer contexts that impose definite, although variable, requirements on the engineering task.

For example, a requirements specification may specify that a given system must be portable across three specific hardware platforms. In a full-scale domain engineering problem as defined within ODM, the selection of the specific platforms to support (and how many to support) would be part of the engineering task. See also variability modeling, variability reengineering.

### variability modeling

The problem of how to represent variability both within and across systems. It includes the problem of distinguishing the binding sites of various functions within both system in development and system in use contexts.

For example, some object-oriented programming languages can express run-time polymorphism in data structures and procedures. Languages like Ada83 use generics that are processed largely at compile-time (simplified for the sake of example); code generators can be written to produce optimized procedures in languages that support no polymorphism at all. Variability modeling would address the problem of how to represent these (and possibly other) design options in a single, integrated design space. Variability engineering would address the problem of deciding which option to use. See also variability engineering.

### variability reengineering

Application and extension of systems reengineering techniques to multiple-system contexts, such as reengineering a product-line. As in variability(forward) engineering, the term implies that there is an established set of customer contexts that impose definite, although variable, requirements on the engineering task. See also variability engineering.

### variant

Two exemplar artifacts are variants if they are classified as instances of a common category but have distinguishing features. The term can also be applied to features themselves, although this implies particular capabilities in the taxonomic modeling representations used. See also architectural variant.

### vertical domain

Domains that include entire systems in their scope. These systems may be closely related (e.g., a set of system versions or a product line) or loosely related (a family of systems). In ODM the terms *vertical* and *horizontal* are not absolute qualities of general functional areas, but describe the span of a given domain's coverage relative to a particular set of systems. For example, within a family of large-scale systems, database management may be considered a horizontal domain. In the Database Management Systems provider marketplace, database management could be considered a vertical domain. See horizontal domain.

### visibility

A practitioner has visibility to a feature in a context if his or her work is affected by the value or variant of the feature in a way that can be attributed to the feature by the practitioner. For example, an end-user may not have visibility to the data representation for an outline document unless restrictions in the implementation make this representation "show through." Thus the scope of visibility is narrower than the potential scope of impact for the feature, and narrower than the scope of control of a given stakeholder.

### workproduct

A textual or graphical document, or a model, produced during a domain engineering project task that exists in persistent form. In this document, this term is used *only* for ODM workproducts. All documents, system models, code modules, etc. from representative systems are considered artifacts from the standpoint of ODM, even though within the context of the application projects themselves the same documents would be considered workproducts. By analogy, if researchers were to collect samples of ODM workproducts from multiple domain engineering projects and compare them for commonality and variability, the documents would be playing the role of artifacts. See also composite workproduct, component workproduct.

# Appendix C: ODM Workproducts

This appendix contains descriptions of the ODM workproducts in Section C.1, followed by templates for a selected set of workproducts in Section C.2.

## C.1 Workproduct Definitions

### ARTIFACTS LIST

See ARTIFACTS LIST under DOMAIN DOSSIER.

### ASSET BASE

The full set of ASSETS for a domain, together with the ASSET BASE ARCHITECTURE that integrates the ASSETS and ASSET BASE INFRASTRUCTURE required for the domain. The ASSET BASE also includes the ASSET BASE MODEL and ASSET SPECIFICATIONS. The ASSET BASE is developed during the Asset Base Engineering phase.

### ASSET BASE ARCHITECTURE

The ASSET BASE ARCHITECTURE models the range of variability in system architectures obtainable from the ASSET BASE. The architecture includes separately selectable asset ensembles. It should be as implementation-neutral as possible. The ASSET BASE ARCHITECTURE is developed in the *Define Asset Base Architecture* task in the *Asset Base Engineering* phase.

### ASSET BASE CUSTOMER MODEL

See ASSET BASE CUSTOMER MODEL under ASSET BASE MODEL.

### ASSET BASE DOSSIER

A composite workproduct consisting of the CUSTOMER CONTEXT DOSSIER and DOMAIN DOSSIER.

### ASSET BASE FEATURE — CUSTOMER MAP

See ASSET BASE FEATURE — CUSTOMER under ASSET BASE MODEL.

### ASSET BASE FEATURE MODEL

See ASSET BASE FEATURE MODEL under ASSET BASE MODEL.

### ASSET BASE INFRASTRUCTURE

Any portion of the environments, tools, documentation, and procedures required for ongoing integrated management of the ASSET BASE that are most effectively developed in tandem with the specific ASSETS of the ASSET BASE. These may include:

- Domain-specific extensions to general infrastructure mechanisms and procedures selected.
- Technology-specific extensions to the ASSET BASE ARCHITECTURE, reflecting additional constraints and semantics imposed by technology choices on the architecture.
- Inputs to the Technology Transfer Plan. The Technology Transfer Plan is developed outside the domain engineering life cycle, in Asset Base Management. Asset Base Management receives and transitions the ASSET BASE to ASSET BASE users.

The ASSET BASE INFRASTRUCTURE is developed in the *Implement Infrastructure* task in the Asset Base Engineering phase.

### ASSET BASE MODEL

Developed in the *Select Features and Customers* task in the Asset Base engineering phase, the ASSET BASE MODEL includes:

- ASSET BASE FEATURE — CUSTOMER MAP: The mapping of selected features to selected customers. This is a refinement of the PRIORITIZED FEATURE — CUSTOMER MAP produced in the *Prioritize Features and Customers* task.
- ASSET BASE FEATURE MODEL: The set of features to be supported by the ASSET BASE.
- ASSET BASE CUSTOMER MODEL: Asset utilizers (customers) characterized by attributes of their context of application, profile (e.g., experience level, role in various life cycle phases, etc.).

### ASSET DELIVERY MECHANISM

See ASSET DELIVERY MECHANISM under ASSETS.

### ASSET IMPLEMENTATION

See ASSET IMPLEMENTATION under ASSETS.

### ASSET IMPLEMENTATION PLAN

Details of the technology to be used and the implementation schedule for each ASSET. The ASSET IMPLEMENTATION PLAN is created in the *Plan Asset Base Implementation* task in the Asset Base Engineering phase and includes the following components:

- PROJECT — CUSTOMER TIME LINE. A template of this workproduct is include in Exhibit C-18.

### ASSET INTERFACE SPECIFICATION

See ASSET INTERFACE SPECIFICATION under ASSETS.

### ASSETS

Workproducts of the software life cycle specifically engineered for reuse within a domain-specific scope of applicability. ASSETS include software components, generative tools, and templates for design documents. Each ASSET has the following structure:

- ASSET INTERFACE SPECIFICATION. Expressed in terms of domain-specific features as documented in the DOMAIN MODEL. The interface specification explicitly documents the intended ***scope of applicability*** for the ASSET, the range of application contexts for which it has been designed.
- ASSET TEST AND VALIDATION PLAN. Developed from the overall TEST AND VALIDATION PLAN, the ASSET TEST AND VALIDATION PLAN provides supplemental documentation of the contextual assumptions and semantics of operation for a single ASSET.
- ASSET IMPLEMENTATION. Can take the form of a software component, a parameterized template, a generative tool that produces an ***asset instance*** at the point of demand, or any of a variety of other forms.
- ASSET DELIVERY MECHANISM. Provides any needed support for delivering the ASSET instan-

tiation to the customer at the point of demand. This can include documentation of suggested integration, validation, and tailoring strategies.

- ASSET SUPPORTING MATERIAL. Other supporting material for an ASSET, possibly including example applications, and a list of potential customer sites derived from the ASSET BASE DOSSIER.

ASSETS are developed in the *Implement Assets* task in the Asset Base Engineering phase.

**ASSET SPECIFICATIONS**

ASSET SPECIFICATIONS contain profiles for individual assets that implement the selected features. The ASSET SPECIFICATIONS include the required range of variability in the ASSETS, and constraints and interdependencies with other ASSETS. The ASSET SPECIFICATIONS are developed in the *Define Asset Base Architecture* task in the Asset Base Engineering phase.

**ASSET SUPPORTING MATERIAL**

See ASSET SUPPORTING MATERIAL under ASSETS.

**ASSET TEST AND VALIDATION PLAN**

See ASSET TEST AND VALIDATION PLAN under ASSETS.

**CANDIDATE DOMAINS MATRIX**

See CANDIDATE DOMAINS MATRIX under DOMAIN SELECTION.

**CANDIDATE EXEMPLAR SYSTEMS**

Indicates SYSTEMS OF INTEREST that intersect the domain of focus, characterized with respect to the domain of focus. Includes all candidate systems considered as potential exemplars for the domain. This set is extracted from the SYSTEMS OF INTEREST by selecting systems identified as potential exemplar systems, based on the informal initial domain understanding that is later formalized in the DOMAIN DEFINITION STATEMENT. The extracted systems are then characterized by creating a DEFINING FEATURES MAP. A template of the DEFINING FEATURES MAP is illustrated in Exhibit C-6. The CANDIDATE EXEMPLAR SYSTEMS are selected in the *Select Domain of Focus* task in the Planning phase.

**CANDIDATE FEATURE — CUSTOMER MAP**

Shows the anticipated ***degree of interest*** that candidate customers have for candidate feature sets derived from the DOMAIN MODEL. Each feature set is a subset of the features in the DOMAIN MODEL. Each customer market is a segment of the DOMAIN STAKEHOLDER MODEL Both elements are still *candidates* only. Final selections are made in subsequent tasks.

The mapping is *many-to-many*: Each feature set may be deemed of interest to multiple customers; this is, in fact, a good approximation of the conventional notion of “reusability” in this context. Conversely, each customer may be interested in multiple feature sets.

The CANDIDATE FEATURE — CUSTOMER MAP is developed in the *Correlate Features and Customers* task in the Asset Base Engineering phase. A template for this workproduct with more detailed procedures is included in Exhibit C-13.

**CANDIDATE PROJECT OBJECTIVES**

A list of candidate objectives for the domain engineering project mapped to stakeholder interests and the PROJECT CHARTER, qualified as positive, negative, or neutral. The CANDIDATE PROJECT OBJECTIVES are developed in the *Identify Candidate Project Objectives* task in Planning. A template of this workproduct is illustrated in Exhibit C-2.

**CANDIDATE PROJECT STAKEHOLDER MODEL**

A list of candidate project stakeholders, partitioned into key and potential stakeholders annotated with descriptions of roles, relationships, interests, enablers and barriers, and assessments of attitudes towards reuse and reuse practices. The CANDIDATE PROJECT STAKEHOLDER MODEL is developed in the *Determine Candidate Project Stakeholders* task in Planning and forms the superset of the analogous model for any particular project or domain within the organization. See PROJECT STAKEHOLDER MODEL, DOMAIN STAKEHOLDER MODEL. A template for defining PROJECT STAKEHOLDER ROLES is illustrated in Exhibit C-1.

**CANDIDATE REPRESENTATIVE SYSTEMS**

See CANDIDATE REPRESENTATIVE SYSTEMS under REPRESENTATIVE SYSTEMS SELECTION.

**COMPONENT CONSTRAINTS MAP**

See COMPONENT CONSTRAINTS MAP under INTERNAL ARCHITECTURE CONSTRAINTS.

**CONCEPTUAL — HISTORICAL RELATIONS MAP**

See DOMAIN INTERCONNECTION MODEL.

**CONTEXT MAP**

See CONTEXT MAP under DOMAIN DOSSIER.

**CUSTOMER CONTEXT DOSSIER**

For those interfaces identified in the DOMAIN INTERCONNECTION MODEL, and those customer systems that were originally part of the REPRESENTATIVE SYSTEMS SELECTION studied during the *Model Domain* phase, relevant features are carried over from the DOMAIN DOSSIER into the CUSTOMER CONTEXT DOSSIER. Any other information about customer context is captured in this dossier as well. The CUSTOMER CONTEXT DOSSIER is created in the *Prioritize Features and Customers* task in the Asset Base Engineering phase. It is used in determining the external and internal constraints on the ASSET BASE ARCHITECTURE in the *Architect Asset Base* sub-phase.

**CUSTOMER — EXTERNAL INTERFACE MAP**

See CUSTOMER — EXTERNAL INTERFACE MAP under EXTERNAL ARCHITECTURE CONSTRAINTS.

**DATA ACQUISITION PLAN**

Plan containing the information sources to be investigated for the domain which is produced in the *Plan Data Acquisition* task in the Domain Modeling phase. The DATA ACQUISITION PLAN includes:

- INFORMATION ACCESSIBILITY — QUALITY MAP. Candidate representatives characterized by information accessibility and quality for each system context. A template of this workproduct is illustrated in Exhibit C-11.

- DOMAIN CONTEXTS MAP. Contains the set of system contexts to be studied for each representative system. These contexts are linked by a mapping that relates contexts of similar types (e.g., all system in development contexts, system in use contexts, etc.) and indicates their domain-wide relationships.
- PRIORITIZED INFORMATION TYPES. Selected representative systems mapped to a refined list of desired information types to be examined. This workproduct is an annotated refinement of the INFORMATION ACCESSIBILITY — QUALITY MAP. For each representative system, the PRIORITIZED INFORMATION TYPES includes:
  — A list of domain contexts to be examined including scoping by context type (e.g., exclude maintenance environments for all representatives).
  — A list of types of artifacts, stakeholder roles, and processes to be studied in each type of context.
  — An indication of whether data to be acquired is representative, sampled or exhaustive.

**DEFINING FEATURES MAP**

See CANDIDATE EXEMPLAR SYSTEMS.

**DESCRIPTIVE MODELS**

A composite workproduct consisting of all of the models produced during the *develop descriptive models* sub-phase of Domain Modeling. The DESCRIPTIVE MODELS include the DOMAIN MODEL LEXICON, DOMAIN CONCEPTS MODELS, and DOMAIN FEATURE MODELS.

**DESCRIPTIVE MODEL TYPES**

See DESCRIPTIVE MODEL TYPES under DOMAIN MODEL ONTOLOGY.

**DOMAIN ATTRIBUTE DEFINITIONS**

See DOMAIN ATTRIBUTE DEFINITIONS under DOMAINS OF INTEREST.

**DOMAIN CONCEPT MODELS**

DOMAIN CONCEPT MODELS are models of the commonality and variability for particular artifacts or objects across all the system contexts of all representatives in which this artifact or object occurs. An ***object*** is a real-world entity represented by data or by an artifact within a system context. DOMAIN CONCEPT MODELS can also describe processes (operations, human actions) or particular stakeholders within domain contexts. The DOMAIN CONCEPT MODELS are created in the *Develop Concept Models* task in the Domain Modeling phase.

For example, in the Outlining Domain, the notion of a "heading" would occur within the concepts model. The DOMAIN CONCEPT MODELS show semantic relationships between domain terms.

**DOMAIN CONTEXTS MAP**

See DOMAIN CONTEXTS MAP under DATA ACQUISITION PLAN.

**DOMAIN DEFINING FEATURES**

See DOMAIN DEFINING FEATURES under INTENSIONAL DOMAIN DEFINITION.

**DOMAIN DEFINITION**

A composite workproduct consisting of the INTENSIONAL DOMAIN DEFINITION, EXEMPLAR SYSTEMS SELECTION, and DOMAIN INTERCONNECTION MODEL. The DOMAIN DEFINITION is created in the *Define Domain* sub-phase in the Asset Base Engineering phase.

**DOMAIN DEFINITION LEXICON**

See DOMAIN DEFINITION LEXICON under INTENSIONAL DOMAIN DEFINITION.

**DOMAIN DEFINITION STATEMENT**

See DOMAIN DEFINITION STATEMENT under INTENSIONAL DOMAIN DEFINITION.

**DOMAIN DOSSIER**

Produced in the *Examine Artifacts* and *Elicit Informant Data* tasks in the Domain Modeling phase. The DOMAIN DOSSIER includes:

- ARTIFACTS LIST: List of artifacts examined, including those called for by the DATA ACQUISITION PLAN and those selected serendipitously. Artifacts can include primary as well as other data (i.e., artifacts not tied to an individual representative system).
- For each representative system:
  - A CONTEXT MAP for each context to be studied, with linkage for artifacts, processes, and practitioners active in that context; e.g., a design document is linked to the context in which it was produced.
  - A SYSTEM CONTEXTS MAP showing detailed linkage between the contexts; information and product flow. This information is largely suggested by the context types (e.g., system in development, system in use contexts) but more details can emerge from data acquisition.
- INFORMAL FEATURE SETS: Sets of ***informal features.*** Includes the author of each feature (e.g., a member of the modeling team inspecting a design, a comment noted during a walk-through with a designer, a comment drawn from a design rationale document). These include:
  - Informal features found in each representative system for the system as a whole and for particular artifacts.
  - Informal features elicited from informants.
- INFORMANT DATA. Reports from interviews with practitioners, observation of work practices, instrumented process capture, etc.

**DOMAIN FEATURE MODELS**

Models of distinguishing characteristics of specific artifacts or representative systems as a whole. Each feature is an assertion or statement that holds for the associated system entity and indicates a significant differentiation from the standpoint of some domain practitioner.

The feature models are developed using the same taxonomic modeling techniques used for the DOMAIN CONCEPT MODELS. Each feature is mapped to:

- associated concepts in the DOMAIN CONCEPTS MODEL;
- via concepts, to the DOMAIN MODEL LEXICON;
- via documented interests, to practitioners in the DOMAIN STAKEHOLDER MODEL.

The DOMAIN FEATURE MODELS are developed in the *Develop Feature Models* task in the Domain Modeling phase. See also DOMAIN CONCEPTS MODEL, DOMAIN MODEL LEXICON, DOMAIN STAKEHOLDER MODEL.

**DOMAIN INFORMANT MODEL**

The DOMAIN INFORMANT MODEL includes the INFORMANT ROLES IN CONTEXT, a map of who knows what about the domain, i.e., informants mapped to representative systems about which they have knowledge and experience. A template of the INFORMANT ROLES IN CONTEXT is illustrated in Exhibit C-12. Also includes the practitioner roles which potential informants have played in each relevant system context. Informants not associated with specific representative systems (e.g., experts with experience with multiple systems) are ranked in a similar way. The Domain Informant Model is created during the *Plan Data Acquisition* task in the Domain Modeling phase.

**DOMAIN INTERCONNECTION MODEL**

Model of the relationship between the selected domain (domain of focus) and various related domains. The DOMAIN INTERCONNECTION MODEL includes:

- DOMAIN AFFINITY DIAGRAM. An informal graphic depiction of the domain of focus in the center of numerous overlapping related domains. It is used as a preliminary way of capturing candidate domains before they are classified more formally.
  - RELATED DOMAINS — DEFINING FEATURES MAP. Used to characterize and classify related domains in terms of the DOMAIN DEFINING FEATURES from the INTENSIONAL DOMAIN DEFINITION. A template of this workproduct is illustrated in Exhibit C-7.
  - Operational/Structural Contexts
  - Life Cycle Contexts
  - Application Contexts
- CONCEPTUAL — HISTORICAL RELATIONS MAP. Classify related domains into specialization domains, generalization domains, and analogy domains. A template of the CONCEPTUAL — HISTORICAL RELATIONS MAP is illustrated in Exhibit C-8.

The DOMAIN INTERCONNECTION MODEL is produced during the *Orient Domain* task of the Planning phase. The DOMAIN INTERCONNECTION MODEL is used to identify major domain variants in later descriptive modeling, possible innovative feature combinations during the *Extend Domain Model* task, and, in general, to ensure a well-bounded domain focus.

**DOMAIN LEXICON**

Extends and refines the DOMAIN DEFINITION LEXICON with terms that have specific meanings for practitioners in the system contexts studied during Domain Modeling (as deduced from the artifacts). The lexicon links domain terms to their occurrence either within specific REPRESENTATIVE SYSTEM ARTIFACTS or as recorded during interviews with specific informants. Syntactic relations such as synonyms are noted where they are clearly suggested by the data in the domain.

The DOMAIN LEXICON is a terminology map that captures and centralizes the specific language of the domain. Since domains often cross organization boundaries, differences in domain informants' use of terms must be reconciled, as well as differences between informants' and modelers' terminology.

The Domain Lexicon is developed during the *Examine Artifacts* and *Elicit Informant Data* tasks during the Domain Modeling phase. See also DOMAIN DEFINITION LEXICON and DOMAIN MODEL LEXICON.

**DOMAIN MODEL**

A composite workproduct consisting of the INTERPRETIVE DOMAIN MODEL and the EXTENDED DOMAIN MODEL. The DOMAIN MODEL provides a formalized language for specifying the intended range of applicability of the ASSETS. The DOMAIN MODEL is created in the *Refine Domain Model* sub-phase in the Domain Modeling phase.

**DOMAIN MODEL INTERCONNECTION MAP**

See DOMAIN MODEL INTERCONNECTION MAP under DOMAIN MODEL ONTOLOGY.

**DOMAIN MODEL LEXICON**

Contains the canonic terms from the DOMAIN LEXICON for each concept requiring differentiation in the domain. The DOMAIN MODEL LEXICON explains domain modelers' terms to the community. Each canonic term is mapped to other domain lexicon terms. Documented rationale for the choice of canonic terms is optional. The DOMAIN MODEL LEXICON is created in the *Develop Lexicon* task in the Domain Modeling phase.

**DOMAIN MODEL ONTOLOGY**

A model that serves as a plan for what specific models to develop during the *Develop Descriptive Models* sub-phase of the Domain Modeling phase. The specific models are loosely related semantically. Each specific model is named and its ***ontology*** specified, that is, the types of instances the model is intended to encompass. The DOMAIN MODEL ONTOLOGY is developed during the *Select Descriptive Model Types* task. It includes:

- DESCRIPTIVE MODEL TYPES. A list of DESCRIPTIVE MODELS to be developed. Used by modelers in allocating modeling tasks, filtering data, and model development. Each model type includes the following attributes:
  - — Name of model to be developed;
  - — Derivation/classification of model type (general, domain-specific);
  - — Types of entities that are the focus of the model;
  - — Rationale for why the model will be developed.
- DOMAIN MODEL INTERCONNECTION MAP. A ***meta-model*** that formalizes selected models in the modeling representation of choice and describes the interconnections between models to be developed.

**DOMAIN MODEL RATIONALE**

Documents process information, rationale, decision histories, trade-offs, and issues surrounding elements in the primary DESCRIPTIVE MODELS. The intent is to develop rationale explaining the commonality and variability in the DESCRIPTIVE MODELS. While usually rationale is specified at the level of the INTEGRATED DOMAIN MODEL, in principle such contextual information could also be recorded about finer-grained conceptual entities such as objects and operations (e.g., when might a particular operation be useful, what are its performance penalties). The DOMAIN MODEL RATIONALE is developed in the *Interpret Domain Model* task in the Domain Modeling phase.

**DOMAIN SELECTION**

A report created in the *Select Domain of Focus* task that documents the activities that lead to selection of a domain of focus. The DOMAIN SELECTION includes:

- The CANDIDATE DOMAINS MATRIX, which characterizes candidate domains by DOMAIN

SELECTION CRITERIA. A template of this workproduct is illustrated in Exhibit C-5.

- A description of the SELECTED DOMAIN(S) of focus and rationale.
- An INFORMAL DOMAIN DEFINITION STATEMENT for domain of focus.

**DOMAIN SELECTION CRITERIA**

A list of prioritized selection criteria that will be used to select the domain of focus, with annotations explaining the derivation of each criterion. The DOMAIN SELECTION CRITERIA are developed in the *Define Selection Criteria* task.

**DOMAINS OF INTEREST**

The DOMAINS OF INTEREST workproduct is created in the *Characterize Domains of Interest* task. Purposes of the DOMAINS OF INTEREST workproduct include:

- Candidate domains in the DOMAINS OF INTEREST are input to selecting the domain of focus.
- Input to the *Orient Domain* task, used in developing the DOMAIN INTERCONNECTION MODEL.
- Reused in subsequent domain engineering projects as a starting point for domain selection. If the ORGANIZATION CONTEXT is stable, the DOMAINS OF INTEREST should change relatively slowly (e.g., only if workers develop expertise in new areas, or if existing areas are partitioned into new domains).

The DOMAINS OF INTEREST workproduct includes:

- DOMAINS OF INTEREST (LIST). Annotated list of DOMAINS OF INTEREST. Useful information that cannot easily be associated with only one domain of interest should also be captured.
- Domain Attribute Definitions
- DOMAINS OF INTEREST CHARACTERIZATION. Matrix characterizing the selected set of domains of interest according to the selected attributes.
- DOMAIN TO SYSTEM MAP. Many-to-many mapping of DOMAINS OF INTEREST to SYSTEMS OF INTEREST. A system is an exemplar of a domain if the domain functionality occurs within the system scope. The DOMAIN TO SYSTEM MAP characterizes domains by their exemplar systems. A template of this workproduct is illustrated in Exhibit C-4.
- DOMAINS OF INTEREST LEXICON. This initial lexicon contains stakeholder terms that refer to principles for grouping families of systems together, or for subsets of functionality within systems or across the life cycle of systems.

**DOMAINS OF INTEREST CHARACTERIZATION**

See DOMAINS OF INTEREST CHARACTERIZATION under DOMAINS OF INTEREST.

**DOMAINS OF INTEREST LEXICON**

See DOMAINS OF INTEREST LEXICON under DOMAINS OF INTEREST.

**DOMAIN SPECIFIC PROJECT OBJECTIVES**

See DOMAIN SPECIFIC PROJECT OBJECTIVES under PROJECT OBJECTIVES.

**DOMAIN STAKEHOLDER MODEL**

A refinement of the PROJECT STAKEHOLDER MODEL which only includes stakeholders with an interest in the domain of focus. The DOMAIN STAKEHOLDER MODEL can also include new stakeholders with interest in the domain, based on the INFORMAL DOMAIN DEFINITION STATEMENT. The DOMAIN STAKEHOLDER MODEL also includes the specific roles of these stakeholders with respect to the domain of focus. The DOMAIN STAKEHOLDER MODEL is developed during the *Select Domain of Focus* task in Planning.

**DOMAIN TO SYSTEM MAP**

See DOMAIN TO SYSTEM MAP under DOMAINS OF INTEREST.

**EXEMPLAR SYSTEMS SELECTION**

A set of example systems classified in terms of domain membership (i.e., exemplars, counter-exemplars, borderline exemplars). This list complements the *intensional* rules contained in the DOMAIN DEFINING FEATURES MODEL. The EXEMPLAR SYSTEMS SELECTION defines the domain by identifying systems judged to be members of the domain. Exemplar systems are chosen from CANDIDATE EXEMPLAR SYSTEMS in the *Bound Domain* task of the Planning phase. See exemplar system, counter-exemplar, borderline exemplar, CANDIDATE EXEMPLAR SYSTEMS.

**EXTENDED DOMAIN MODEL**

A transformation of the INTERPRETIVE DOMAIN MODEL created using model innovation strategies to discover novel combinations and permutations of existing features and architectural variants. Variant features within this model may have no known precedent within the exemplar systems.

- EXTENDED FEATURE MODEL: Contains both precedented and unprecedented features and innovative extensions to features. The representation form used should be compatible with those used in earlier sub-phases. The content will be a superset of the INTERPRETIVE DOMAIN MODEL.
- EXTENDED STAKEHOLDER MODEL: Contains new semantic information about domain stakeholders, including attributes of potential application contexts for features in the EXTENDED FEATURE MODEL.

The EXTENDED DOMAIN MODEL is produced during the *Extend Domain Model* task of the Domain Modeling phase.

**EXTENDED FEATURE MODEL**

See EXTENDED FEATURE MODEL under EXTENDED DOMAIN MODEL.

**EXTENDED STAKEHOLDER MODEL**

See EXTENDED STAKEHOLDER MODEL under EXTENDED DOMAIN MODEL.

**EXTERNAL ARCHITECTURE CONSTRAINTS**

Developed in the *Determine External Architecture Constraints* task in the Asset Base Engineering phase, the EXTERNAL ARCHITECTURE CONSTRAINTS include:

- KEY EXTERNAL INTERFACES. A list of the interfaces to ASSET BASE functionality to be supported, for each domain context for which reusable ASSETS are to be developed. Interfaces include environmental and sub-system interfaces. Environmental and sub-system interfaces can be partitioned if appropriate, but it is possible that the determination of which category an interface falls into may depend on final architectural decisions that will be decided in a later task.
- CUSTOMER — EXTERNAL INTERFACE MAP. A matrix documenting relevant constraints for each customer in the ASSET BASE MODEL has on the form of each external interface. A tem-

plate of this workproduct is illustrated in Exhibit C-12.

- FEATURE — EXTERNAL INTERFACE MAP. Allocation of features from the ASSET BASE FEATURE MODEL to the interfaces identified in each customer context. A template of this workproduct is illustrated in Exhibit C-14.

**FEATURE BINDING SITES MAP**

See FEATURE BINDING SITES MAP under INTEGRATED DOMAIN MODEL

**FEATURE ENSEMBLE CONSTRAINTS**

See FEATURE ENSEMBLE CONSTRAINTS under INTERNAL ARCHITECTURE CONSTRAINTS.

**FEATURE ENSEMBLE — CUSTOMER CONTEXT MAP**

See FEATURE ENSEMBLE — CUSTOMER CONTEXT MAP under INTERNAL ARCHITECTURE CONSTRAINTS.

**FEATURE — EXTERNAL INTERFACE MAP**

See FEATURE — EXTERNAL INTERFACE MAP under EXTERNAL ARCHITECTURE CONSTRAINTS.

**INFORMAL DOMAIN DEFINITION STATEMENT**

See INFORMAL DOMAIN DEFINITION STATEMENT under DOMAIN SELECTION.

**INFORMAL FEATURE SETS**

See INFORMAL FEATURE SETS under DOMAIN DOSSIER.

**INFORMANT DATA**

See INFORMANT DATA under DOMAIN DOSSIER.

**INFORMANT ROLES IN CONTEXT**

See DOMAIN INFORMANT MODEL.

**INFORMATION ACCESSIBILITY — QUALITY MAP**

See INFORMATION ACCESSIBILITY — QUALITY MAP under DATA ACQUISITION PLAN.

**INFRASTRUCTURE IMPLEMENTATION PLAN**

Contains a plan for the implementation of an ASSET BASE INFRASTRUCTURE. The INFRASTRUCTURE IMPLEMENTATION PLAN includes technology selections and scheduling constraints for infrastructure development relative to development of ASSET releases. The infrastructure may be specifically developed for the ASSET BASE or may be available technology that is applied to the ASSET BASE. In general, individual ASSET IMPLEMENTATION PLANS will be affected by the choice of ASSET BASE INFRASTRUCTURE. The INFRASTRUCTURE IMPLEMENTATION PLAN is created in the *Plan Asset Base Implementation* task in the Asset Base Engineering phase.

**INTEGRATED DOMAIN MODEL**

The unified and integrated model of feature variants synthesized from the individual DOMAIN FEATURE MODELS during the *Integrate Descriptive Models* task. This model includes semantic relationships between features, based where possible on implications of the semantics defined on the underlying descriptive artifact, conceptual and contextual models. The INTEGRATED DOMAIN MODEL is developed in the *Integrate Descriptive Models* task in the Domain Modeling phase. It includes:

- FEATURE BINDING SITES MAP: integrates stakeholders with interests in features from the DOMAIN STAKEHOLDERS MODEL, domain contexts, and associated roles of practitioners with visibility/access to features, from the DOMAIN CONTEXTS MODEL. The FEATURE BINDING SITES MAP refines these relationships down to the next level of granularity, to the individual feature binding sites within each domain context and their relationships to practitioner roles.
- INTEGRATED FEATURE MODEL: A linked version of the separate DOMAIN FEATURE MODELS with redundancies and inconsistencies removed.
- REPRESENTATIVE SYSTEMS FEATURE PROFILE: A profile of the REPRESENTATIVE SYSTEMS SELECTION in terms of the INTEGRATED FEATURE MODEL.

## INTEGRATED FEATURE MODEL

See INTEGRATED FEATURE MODEL under INTEGRATED DOMAIN MODEL.

## INTENSIONAL DEFINITION TO INTERCONNECTION MODEL MAP

See DOMAIN INTERCONNECTION MODEL.

## INTENSIONAL DOMAIN DEFINITION

Produced in the *Bound Domain* task in the Planning phase. The INTENSIONAL DOMAIN DEFINITION includes:

- DOMAIN DEFINITION STATEMENT. Includes the name of the domain. The name should be enriched with each decision about domain bounding criteria, although it may not be possible to convey all the criteria adequately in the name. The DOMAIN DEFINITION STATEMENT serves the project team as a consensus document about what the domain includes. In addition, the DOMAIN DEFINITION STATEMENT communicates to domain practitioners the specific scope addressed by the domain of focus. The definition can be targeted to experts or novices in the domain.
- DOMAIN DEFINITION LEXICON. Contains a subset of terms from DOMAINS OF INTEREST LEXICON relevant to the selected domain. Also contains terms elicited from domain stakeholders and information sources that describe defining features for the domain.
- DOMAIN DEFINING FEATURES. Contains the defining features of the domain in terms of rules of inclusion and exclusion and the logical relationship between these rules. A ***defining rule*** is a statement about various combinations of features that imply membership in the domain. A template of this workproduct is illustrated in Exhibit C-6.

## INTERNAL ARCHITECTURE CONSTRAINTS

Developed in the *Determine Internal Architecture Constraints* task in the Asset Base Engineering phase, the INTERNAL ARCHITECTURE CONSTRAINTS include:

- FEATURE ENSEMBLE — CUSTOMER CONTEXT MAP. Documents which feature ensembles are *desirable* in contexts of use and which feature ensembles are *feasible* in contexts of development. Refines the information in the ASSET BASE MODEL.
- COMPONENT CONSTRAINTS MAP. A map of the feature ensembles that can potentially be selected as stand-alone or separately selectable. A template of a COMPONENT CONSTRAINTS MAP workproduct is illustrated in Exhibit C-9.

- LAYERING CONSTRAINTS MAP. A map of subsets/layered architecture variants to feature ensembles. A template of a LAYERING CONSTRAINTS MAP workproduct is illustrated in Exhibit C-10.
- FEATURE ENSEMBLE CONSTRAINTS. Separate selectability constraints and other interactions among ensembles.

**INTERPRETIVE DOMAIN MODEL**

INTEGRATED DOMAIN MODEL extended with:

- Features clusters with strong rationale for co-occurrence.
- Some feature occurrences documented as historical "vestiges" rather than strongly correlated with contextual profiles.
- Feature constraints, which effectively reduce the possible feature space to exclude combinations interpreted as out of scope or semantically meaningless.
- Contextual model describing attributes of domain practitioners or surrounding system context that are relevant to the explanatory model of features derived in this task.

The INTERPRETIVE DOMAIN MODEL is developed in the *Interpret Domain Model* task of the Domain Modeling phase.

**KEY EXTERNAL INTERFACES**

See KEY EXTERNAL INTERFACES under EXTERNAL ARCHITECTURE CONSTRAINTS.

**LAYERING CONSTRAINTS MAP**

See LAYERING CONSTRAINTS MAP under INTERNAL ARCHITECTURE CONSTRAINTS.

**NEW SYSTEM ARTIFACTS**

NEW SYSTEM ARTIFACTS include any of the artifacts for representative systems that were prototyped, reverse or "lateral" engineered to fill gaps in the data. These workproducts can be by-products of the *Acquire Domain Information* task.

Because these artifacts may or may not be clearly associated with a single representative system they are termed only "**system** artifacts". Because the artifacts are created in the course of domain engineering, rather than existing artifacts that are examined, described and annotated, they are "**new** system artifacts". Because they are created as data for domain engineering, rather than to build any individual systems, they are considered **artifacts** rather than workproducts.

**PRIORITIZED FEATURE — CUSTOMER MAP**

Usability and feasibility estimate for each feature set and correlated market segment. The PRIORITIZED FEATURE — CUSTOMER MAP is developed in the *Prioritize Features and Customers* task in the Asset Base Engineering phase. It is a refinement of the FEATURE — CUSTOMER MAP produced in the *Correlate Features and Customers* task.

**PRIORITIZED INFORMATION TYPES**

See PRIORITIZED INFORMATION TYPES under DATA ACQUISITION PLAN.

**PROJECT — CUSTOMER TIME LINE**

See PROJECT — CUSTOMER TIME LINE under ASSET IMPLEMENTATION PLAN.

**PROJECT OBJECTIVES**

A list of the selected project objectives linked to the PROJECT CHARTER and to project stakeholder interests. Objectives are documented clearly and concisely, in language that will easily understood by new project members over the lifetime of the project. The objectives also serve as a statement of expectations for upper management. The PROJECT OBJECTIVES are documented during the *Select Project Objectives and Stakeholders* task in Planning. Domain Specific Project Objectives are added in the Select Domain of Focus task in Planning.

- DOMAIN-SPECIFIC PROJECT OBJECTIVES. Contains specific objectives for the project related to the domain of focus. Also contains potential assets of interest for the domain and potential asset base customer contexts.
- Stakeholder Objectives Map. Links project objectives to project stakeholder interests. A template of this workproduct is illustrated in Exhibit C-3.

**PROJECT — REPRESENTATIVE TIME LINE**

See PROJECT — REPRESENTATIVE TIME LINE under REPRESENTATIVE SYSTEMS SELECTION.

**PROJECT STAKEHOLDER MODEL**

The PROJECT STAKEHOLDER MODEL contains a list of the stakeholders whose interests will be satisfied by the selected PROJECT OBJECTIVES. Stakeholders are annotated with descriptions of roles, relationships, interests, enablers and barriers, and assessments of attitudes towards reuse and reuse practices. The PROJECT STAKEHOLDER MODEL is created in the *Select Project Objectives and Stakeholders* task in Planning and contains stakeholders from the CANDIDATE PROJECT STAKEHOLDER MODEL who are still of interest to the project and any new stakeholders that emerge as a result of selected objectives. Also described are any stakeholder roles that will change if the domain engineering project is successful.

**PROJECT STAKEHOLDER ROLES**

See CANDIDATE PROJECT STAKEHOLDER MODEL.

**RELATED DOMAINS — DEFINING FEATURES MAP**

See RELATED DOMAINS — DEFINING FEATURES MAP under DOMAIN INTERCONNECTION MODEL.

**REPRESENTATIVE SELECTION CRITERIA**

See REPRESENTATIVE SELECTION CRITERIA under REPRESENTATIVE SYSTEMS SELECTION.

**REPRESENTATIVE SYSTEMS FEATURE PROFILE**

See REPRESENTATIVE SYSTEMS FEATURE PROFILE under INTEGRATED DOMAIN MODEL.

**REPRESENTATIVE SYSTEMS SELECTION**

Produced in the *Plan Data Acquisition* task in the Domain Modeling phase. The REPRESENTATIVE SYSTEMS SELECTION includes:

- CANDIDATE REPRESENTATIVE SYSTEMS. The initial filtered list of candidate exemplar systems.

- REPRESENTATIVE SELECTION CRITERIA. Including general and project-specific criteria.
- PROJECT — REPRESENTATIVES TIME LINE. Indicates the schedule relationships between the domain engineering project data acquisition tasks and representative system project schedules. A template of this workproduct is illustrated in Exhibit C-9.
- SELECTION MATRIX. Maps candidate representatives to selection criteria. A template of this workproduct is illustrated in Exhibit C-10.

**SELECTION MATRIX**

See SELECTION MATRIX under REPRESENTATIVE SYSTEMS SELECTION.

**STAKEHOLDER OBJECTIVES MAP**

See STAKEHOLDER OBJECTIVES MAP under PROJECT OBJECTIVES.

**SYSTEM CONTEXTS MAP**

See SYSTEM CONTEXTS MAP under DOMAIN DOSSIER.

**TEST AND VALIDATION PLAN**

A plan for testing and validating implemented ASSETS. The TEST AND VALIDATION PLAN is created in the *Plan Asset Base Implementation* task in the Asset Base Engineering phase.

## C.2 Workproduct Templates

The workproduct templates below are intended to suggest how specific workproducts can be organized and formatted. In general, they are not usable as-is in any specific context because, for example, the numbers of rows and columns (and associated labels) in most of these tabular workproducts will change depending on the domain, the number of exemplars, the number of stakeholders, etc. Thus, the templates will need to be adapted for use on specific projects.

Each template is preceded by a line identifying the ODM process in which the workproduct is primarily created and the guidebook section number in which that process is described. However, there are a number of instances where one template, suggested for use within a given process, could be easily adapted for use in other processes. An example of this "template reuse" can be seen in the similarity between the PROJECT — REPRESENTATIVE SYSTEM TIMELINE and PROJECT — CUSTOMER TIMELINE templates.

This set of templates is not complete and may contain significant inaccuracies and omissions that will be rectified in the next version of the guidebook. In some cases, the tabular form of the templates may be superceded by richer modeling tools and techniques. Use these templates creatively and not literally to produce a concrete set of workproducts for your project.

5.1.1 *Determine Candidate Project Stakeholders*

| Stakeholders | Roles | | | | |
|---|---|---|---|---|---|
| | Key Roles | | | | Potential Roles |
| | Sponsor | DE | SE | System Practitioner | |
| Key Stakeholder 1 | | | | | |
| Key Stakeholder 2 | | | | | |
| Key Stakeholder 3 | | | | | |
| Key Stakeholder 4 | | | | | |
| Potential Stakeholder 1 | | | | | |
| Potential Stakeholder 2 | | | | | |
| Potential Stakeholder 3 | | | | | |
| Potential Stakeholder 4 | | | | | |

**Exhibit C-1.** Project Stakeholder Roles

**Key:**

DE - Domain Engineer

SE - System Engineer

Cells are checked if the stakeholder plays that role.

Stakeholders can play multiple roles.

Multiple stakeholders can hold one role.

## 5.1.2 *Identify Candidate Project Objectives*

| Stakeholder Interests | Candidate Objectives | | |
|---|---|---|---|
| | Objective 1 | Objective 2 | . . . |
| Key Stakeholder 1 | | | |
| Interest A | ① | | |
| Interest B | | ① | |
| Interest C | ① | ① | |
| Key Stakeholder 2 | | | |
| Interest A | ① | | |
| Interest B | | ① | |
| Interest C | ① | ① | |
| Other Stakeholders | | | |
| | | | |
| | | | |
| | | | |

**Exhibit C-2.** Candidate Project Objectives

**Key:**

① List Stakeholder interests, including incentives and barriers

| | |
|---|---|
| ! | - Must have (non-negotiable) |
| +/S | - Nice to have (supports synergy) |
| 0/N | - Neutral |
| -/C | - Risk (conflicts) |
| X | - Can not have (non-negotiable) |
| ? | - Unknown, uncertain |

## 5.1.3 *Select Project Stakeholders and Objectives*

| | Project Stakeholders (Selected & Prioritized) | Project Objectives (Selected & Prioritized) | | | |
|---|---|---|---|---|---|
| | | Success Criteria | | Quality Criteria | |
| | | Objective 1 | Objective 2 | Objective 1 | Objective 2 |
| Key | Stakeholder 1 | | | | |
| | Stakeholder 2 | | | | |
| | . . . | | | | |
| Other | Stakeholder 1 | | | | |
| | Stakeholder 2 | | | | |
| | . . . | | | | |

**Exhibit C-3.** Stakeholder Objectives Map

**Key:**

+ Good fit
0 Neutral
- High risk

Annotate cells to summarize the impact of the objective for each stakeholder in order to identify incentives and mitigate risk.

## 5.2.1 *Characterize Domains of Interest*

| Domains of Interest | Systems of Interest ① | | | |
|---|---|---|---|---|
| | S1 | S2 | ... | Sn |
| D1 ② | ③ | | | |
| D2 | | | | |
| ... | | | | |
| Dn | | | | |

**Exhibit C-4.** Domain-to-System Map

**Key:**

① Input to the task, established by organization context
② Domains of Interest
③ Cells coded with:
V -Vertical domain with respect to this system
E - Encapsulated horizontal domain
D - Distributed
N/A - Not applicable

## 5.2.3 *Select Domain of Focus*

| Candidate Domains of Focus | | Domain Selection Criteria (weighted) | | | |
|---|---|---|---|---|---|
| | Total | Cr 1 | Cr 2 | . . . | Cr n |
| Candidate Domain 1 | | | | | |
| Candidate Domain 2 | | | | | |
| . . . | | | | | |
| | | | | | |
| | | | | | |
| | | | | | |
| Candidate Domain n | | | | | |

**Exhibit C-5.** Candidate Domains Matrix

## 5.3.2 *Bound Domain*

| Candidate Exemplar Systems | | Defining Features and Rules ② | | | | | |
|---|---|---|---|---|---|---|---|
| | Classify ④ | Rule 1 | Rule 2 | . . . | | | Rule n |
| Candidate Exemplar 1 ① | C | | | | | | |
| Candidate Exemplar 2 | E | | | | | | |
| . . . | . . . | | | | | | |
| | | | ③ | | ③ | | |
| | | | | | | | |
| | | | | ③ | | | |
| | | | | | | | |
| | | | | | | | |
| | | ③ | | | | | |
| | | | | | | | |
| | | | | | | | |
| | | | | | | | |
| Candidate Exemplar n | | | | | | | |

**Exhibit C-6.** Domain Defining Features

**Key:**

① List candidate exemplar systems
② Convert informal definition to defining features and, where possible, rules
③ Select all rules and features for each candidate
④ Classify each candidate: E - Exemplar, C - Counter, B - Borderline

5.3.2 *Orient Domain*

| Related Domains | Defining Features/Rules (refer to columns from Exhibit C-6) | | | | | | Differentiating Features |
|---|---|---|---|---|---|---|---|
| **Generalization**<br>Domain Name ⑤<br>________<br>________ | ✓ | ✓ | ✓ | X | | X<br>① | ② |
| Domain Name ⑤<br>________<br>________ | | | | | | | |
| **Analogy**<br>Domain Name ⑤<br>________<br>________ | ✓ | ✓ | ✓ | X | X | ✓<br>③ | ③ |
| Domain Name ⑤<br>________<br>________ | | | | | | | |
| **Specialization**<br>Domain Name ⑤<br>________<br>________ | ✓ | ✓ | ✓ | ✓ | ✓ | ✓ | ④ |
| Domain Name ⑤<br>________<br>________ | | | | | | | |

**Exhibit C-7.** Related Domains — Defining Features Map

**Key:**

① Generalization domains should have at least one unsatisfied domain defining feature
② Possible but not necessary
③ Analogy should have some non-domain features here
④ Specialization should have all defining features and other features
⑤ Names of possible border or counter exemplars, from Exhibit C-6 #4

5.3.2 *Orient Domain*

This template can be used in two ways:

- to map relationships between candidate exemplars in the domains (i.e., fill the cells with names of candidate border or counter exemplars from the template in Exhibit C-6).
- to map relationships between domains (i.e., fill the cells with names of related domains).

Purpose of the template:

- to correlate conceptual and historical relations to domain of focus.
- to identify long-term evolutionary trends and forecast future directions.

| | Past Systems | Current Systems | Future Systems (Forecasting) |
|---|---|---|---|
| Generalization | Early limited function systems | General platforms | Modularization<br>"Lite" configurations |
| Analogy | Manual precursors<br>Leveraged reuse | Analogies used by:<br>• Designers<br>• Interfacers<br>• User training | Dual use / spin-off |
| Specialization | Custom legacy systems | Tailoring to specific environments<br>Values-added extensions | Layered products<br>Customization |

**Exhibit C-8.** Conceptual — Historical Relations Map

6.1.1 *Plan Data Acquisition*

| Representatives | Plan Data Acquisition | | | | | |
|---|---|---|---|---|---|---|
| | Calendar | | | | | |
| | 1Q96 | 2Q96 | 3Q96 | 4Q96 | 1Q97 | 2Q97 |
| Representative 1 | | R —— | D —— | C —— | T —— | Deploy |
| Representative 2 | | R —— | D —— | | C —— | T ... |
| ... | | | ... | | | |
| Representative n | | | | | | R — |
| Domain Engineering Project Schedule | Acquire Data —— | | Descriptive —— | | | Refine |
| Data Acquisition Milestones | —— | ▲ Dossier 1 —— | ▲ Dossier 2 —— | ▲ Model 1 —— | | ▲ Model 2 — |
| Ranking of Representatives for Data Acquisition | | ↑ Rep 1 | ↑ Rep 2 | ... | ↑ Rep n | |

**Exhibit C-9.** Project — Representatives Time Line

**Key:**

R - Requirements Analysis

D - Design

C - Code

T - Test

6.1.1 *Plan Data Acquisition*

| Candidate Representatives | Criteria for Selecting Representatives | | | | | |
|---|---|---|---|---|---|---|
| | Project Dependent Objectives | Project Constraints | Quality — Accessibility-Informant | Genealogy Information | Other | Status |
| Candidate Rep. 1 | ① | ② | ③ | ④ | | ⑤ |
| Candidate Rep. 2 | | | | | | |
| . . . | | | | | | |
| | | | | | | |
| | | | | | | |
| Candidate Rep. n | | | | | | |

**Exhibit C-10.** Representative Systems Selection: Selection Matrix

**Key:**

① Fit or Problem - Document if problem
② Fit or Problem - List problem constraint
③ Rating 0-10
④ Name other candidates that are related -- document relations if complex
⑤ In/Out/? - If ?, circle problem cell

6.1.1 *Plan Data Acquisition*

| Candidate Representative Systems | Possible Contexts of Study | | | | | | | |
|---|---|---|---|---|---|---|---|---|
| | System-in-Development | | SIx | | SIx | | Systems-in-Use | |
| | Artifacts (A) | Processes (P) | A | P | A | P | A | P |
| Candidate Rep. 1 | H/E | | | | | | | |
| Candidate Rep. 2 | | L/D | | | | | | |
| . . . | | L/L | | | | | | |
| | | H/D | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| | | | | | | | | |
| Candidate Rep. n | | | | | | | | |

**Exhibit C-11.** Information Accessibility — Quality Map

**Key:**
Each cell rates both:

- Quality of Info:
  - H - High
  - L - Low
- Ease of Accessibility:
  - D - Difficult
  - E - Easy

SIx - Other possible contexts

## 6.1.1 *Plan Data Acquisition*

| Informants | Candidate Representative Systems Context | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | Representative 1 | | | Representative 2 | | | Representative 3 | | |
| | S-I-D | S-I-x | S-I-U | S-I-D | S-I-x | S-I-U | S-I-D | S-I-x | S-I-U |
| Informant 1 | | | | | | | ① | | |
| Informant 2 | | | | | | ① | | | |
| Informant 3 | ① | | ① | | | | | | |
| Informant 4 | ① | | | ① | | | | | |

**Exhibit C-12.** Informant Roles in Context

**Key:**

① Role of Informant in the Context, in the Representative Candidate — e.g., Developer, System Manager, Secretary, TQM Person, VP R&D, Technical Writer

| | |
|---|---|
| S-I-D | - System-in-Development |
| S-I-x | - Other possible contexts |
| S-I-U | - System-in-Use |

## 7.1.1 *Correlate Features and Customers*

| Candidate Feature Sets | Candidate Customers | | | |
|---|---|---|---|---|
| | C1 ③ | C2 ③ | ... | Cx ③ |
| ① F1 <Descriptor> | ⑤ | ⑤ | d) | |
| Profile: ② F1A _______ | ④ | ④ | | b) |
| ② F1B _______ | ④ | | | b) |
| .... | a) | a) | | a) |
| ② F1x _______ | | | | b) |
| ① F2 <Descriptor> | ⑤ c) | ⑤ c) | d) | c) |
| Profile: F2A _______ | ④ | | | |
| F2B _______ | | ④ | | |
| ... | | | | |
| F2x _______ | | | | |
| ... | | | | |
| Fx, etc. | | | d) | |

**Exhibit C-13.** Candidate Feature — Customer Map

**Key:**

① Initial list of candidate feature sets, tagged by short descriptors
② For each feature set, list each attribute of potential market profile
③ Initial list of candidate customers
④ In the cells:
How close a match is this attribute to this customer?

| | |
|---|---|
| ! | - Close match/direct hit |
| + | - Good match |
| N | - Neutral |
| - | - Poor match/risk |
| X | - Non-match |
| ? | - Unknown/uncertain |

⑤ Summarize match of feature set to each customer

**Notes:**
a) Each attribute can match with multiple customers
b) Each customer can match with multiple attributes
c) A feature set as a whole can match with multiple customers
d) A customer can match with multiple feature sets

## 7.2.1 *Determine External Architecture Constraints*

| Asset Base Feature | Context and Interfaces | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | ② S-I-D Context | | | S-I-x Context | | | S-I-U Context | | |
| | Id1 | Id2 | ... | Ix1 | Ix2 | ... | Iu1 | Iu2 | ... |
| Feature 1 ① | ③ | | | | | | | | |
| Feature 2 | | | | N/A | N/A | N/A | | | |
| ... | | | | | | | | | |
| Feature n | | | | | | | | | |

**Exhibit C-14.** Feature — External Interface Map

**Key:**

① Individual features from ASSET BASE FEATURE MODEL; in prioritized order
② Contexts and interfaces from the ASSET BASE FEATURE MODEL
③ Indicate allocation of features to interfaces

| | |
|---|---|
| S-I-D | - System-in-Development |
| S-I-x | - Other possible contexts |
| S-I-U | - System-in-Use |

7.2.1 *Determine External Architecture Constraints*

| Asset Base Customer | Context and Interfaces | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | ② S-I-D Context | | | S-I-x Context | | | S-I-U Context | | |
| | Id1 | Id2 | . . . | Ix1 | Ix2 | . . . | Iu1 | Iu2 | . . . |
| Customer 1 ① | ③ | | | | | | | | |
| Customer 2 | | | | N/A | N/A | N/A | | | |
| . . . | | | | | | | | | |
| Customer n | | | | | | | | | |

**Exhibit C-15.** Customer — External Interface Map

**Key:**

① Customers from ASSET BASE CUSTOMER MODEL; in prioritized order
② Contexts and interfaces from the ASSET BASE CUSTOMER MODEL
③ In each cell, key constraints on the interface for that customer
N/A if interface and/or context does not apply to that customer

| | |
|---|---|
| S-I-D | - System-in-Development |
| S-I-x | - Other possible contexts |
| S-I-U | - System-in-Use |

## 7.2.2 *Determine Internal Architecture Constraints*

| Feature Ensembles | | Architecture Variability Criteria | | | |
|---|---|---|---|---|---|
| | | Separately Selectable/ Stand-alone | Masking | Internal Optimization | ... |
| Partition 1 ① | Feature Ensemble 1.1 | ② | ② | ② | |
| | Feature Ensemble 1.2 | H/L | L/L | L/H | |
| | ... | | | | |
| | Feature Ensemble 1.n | | | | |
| Partition 2 | Feature Ensemble 2.1 | | | | |
| | Feature Ensemble 2.2 | | | | |
| | ... | | | | |
| | Feature Ensemble 2.n | | | | |
| ... | ... | | | | |
| Partition n | Feature Ensemble n.n | | | | |
| | Feature Ensemble n.n | | | | |
| | ... | | | | |
| | Feature Ensemble n.n | | | | |

**Exhibit C-16.** Component Constraints Map

**Key:**

① Group FEs that partition ASSET FEATURE MODEL
② For each criterion, rate usability and feasibility

| | |
|---|---|
| L | - Low |
| - | - Neutral |
| H | - High |

## 7.2.2 *Determine Internal Architecture Constraints*

| Subsets/ Layered Architecture Variants | Feature Ensembles | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | FE 1 | FE 2 | … | | | FE n | Separate Selectability | Internal Optimization | … |
| Layer 1 ① | X ② | X | | | | | ③ | | |
| Layer 2 | X | | X | X | | | H/L | L/L | |
| Layer 3 | X | X | X | X | | | H/H | H/L | |
| Layer 4 | X | X | | X | | | | | |
| Layer 5 | X | | X | X | | | | | |
| … | | | | | | X | | | |
| Layer n | X | X | X | X | X | X | | | |

**Exhibit C-17.** Layering Constraints Map

**Key:**

① Put candidates, layers or specializations in approximate "subset - first" order
② Indicate which feature ensembles are part of the layer
③ Rate each layering variant for usability and feasibility

| | |
|---|---|
| L | - Low |
| - | - Neutral |
| H | - High |

## 7.3.1 *Plan Asset Base Implementation*

| Customers | Plan Asset Base Implementation | | | | | |
|---|---|---|---|---|---|---|
| | Calendar | | | | | |
| | 1Q96 | 2Q96 | 3Q96 | 4Q96 | 1Q97 | 2Q97 |
| Customer 1 | | R —— | D —— | C —— | T —— | Deploy |
| Customer 2 | | R —— | D —— | C —— | T | ... |
| ... | | | ... | | | |
| Customer n | | | | | | R — |
| Domain Engineering Project Schedule | Plan —— | | Model —— | | Asset —— | —— |
| Deliverables | —— | ▲ —— | ▲ —— | ▲ —— Assets 1 | | ▲ — Assets 2 |
| Ranking of Customer Receptors | | | | ↑ Cust n Cust 2 | | ↑ Cust 1 |

**Exhibit C-18.** Project — Customer Time Line

**Key:**

R - Requirements Analysis

D - Design

C - Code

T - Test

# References

In the following, documents which pertain most directly to ODM and its application are denoted with an asterisk (*).

[ADER95a]* STARS. Army STARS Demonstration Project Experience Report. Unisys STARS Technical Report STARS-VC-A011R/002/02, STARS Technology Center, Arlington VA, February 1995.

[Bail92a] Bailin, S. KAPTUR: Knowledge Acquisition for Preservation of Tradeoffs and Underlying Rationales. CTA Inc., Rockville MD, 1992.

[Bloc81] Block, P. *Flawless Consulting: A Guide to Getting Your Expertise Used.* Pfeiffer & Co., San Diego CA, 1981.

[Boeh92a] Boehm, B., W. Scherlis. "Megaprogramming." In *Proceedings of the DARPA Software Technology Conference*, Arlington VA, April 1992.

[Boji91] Bojie, D. "The Storytelling Organization: A Study of Story Performance in a Office-supply Firm." *Administrative Science Quarterly*, Vol. 36, 1991, pp.106-26.

[Brow91] Brown, J. S. "Research that Reinvents the Corporation." *Harvard Business Review*, March-April 1991.

[CFRP93a]STARS. STARS Conceptual Framework for Reuse Processes (CFRP), Vol. I: Definition, Version 3.0. Unisys STARS Technical Report STARS-VC-A018/001/00, STARS Technology Center, Arlington VA, October 1993.

[CFRP93b]STARS. STARS Conceptual Framework for Reuse Processes (CFRP), Vol. II: Application, Version 1.0. Unisys STARS Technical Report STARS-VC-A018/002/00, STARS Technology Center, Arlington VA, September 1993.

[Clem95a] Clements, P., P. Kogut. "Features of Architecture Description Languages." In *Proceedings of the 7th Annual Software Technology Conference*, Salt Lake City UT, April 1995.

[Coll91a] * Collins, P. "Toward a Reusable Domain Analysis." In *Proceedings of the 4th Annual Workshop on Software Reuse*, Herndon VA, November 1991.

[Crep95a] Creps, R., M.J. Davis, M. Simos, et al. "Using a Conceptual Framework for Reuse Processes as a Basis for Reuse Planning." In *Proceedings of the 7th Annual Software Technology Conference*, Salt Lake City UT, April 1995.

[DEGB95a]* Army STARS Demonstration Project. Domain Engineering Guidebook. U.S. Army CECOM Software Engineering Directorate, Ft. Monmouth NJ, 1995.

[DISA93a]DISA/CIM Software Reuse Program. Domain Analysis and Design Process, Version 1. Technical Report 1222-04-210/30.1, DISA Center for Information Management, Arlington VA, March 1993.

[Fore92a] Foreman, J. "STARS Mission." In *Proceedings of the DARPA Software Technology Conference*, Arlington VA, April 1992.

[Goma90a]Gomaa, H. A Domain Requirements Analysis and Specification Method. Technical Report, George Mason University, Fairfax VA, February 1990.

[Grei72] Greiner, L. “Evolutions and Revolutions as Organizations Grow.” *Harvard Business Review*, July-August 1972.

[IDEF81] Softech, Inc. “Integrated Computer-Aided Manufacturing (ICAM) Architecture Part II. Volume IV - Function Modeling Manual (IDEF0).” Technical Report AFWAL-TR-81-4023 Volume IV, Materials Laboratory (AFWAL/MLTC), AF Wright Aeronautical Laboratories (AFSC), Wright-Paterson AFB, Dayton OH, June 1981.

[Kang90a] Kang, K, S. Cohen, et al. Feature-Oriented Domain Analysis (FODA) Feasibility Study. Technical Report CMU/SEI-90-TR-21, Software Engineering Institute, Pittsburgh PA, November 1990.

[Klin95a] Klingler, C. “A Practical Approach to Process Definition.” In *Proceedings of the 7th Annual Software Technology Conference*, Salt Lake City UT, April 1995.

[Liev80] Lievegoed, B. *The Developing Organization.* Celestial Arts, Millbrae CA, 1980.

[Lind93] Linde, C. "Reflections on Workplace Learning." Working Paper, Institute for Research on Learning, 2550 Hanover St., Palo Alto CA, 1993.

[Marc88] Marca, D., and C. McGowan. *SADT, Structured Analysis and Design Technique.* McGraw-Hill, New York NY, 1988.

[McNi88a]McNicholl, D., S. Cohen, et al. Common Ada Missile Packages - Phase 2 (CAMP-2) - Vol. 1: CAMP Parts and Parts Composition System. Final Report AFAL-TR-88-62, McDonnell Douglas, St. Louis MO, November 1988.

[Moor91a] Moore, G. *Crossing the Chasm: Marketing and Selling Technology Products to Mainstream Customers.* Harper Business, New York NY, 1991.

[Neig83a] Neighbors, J. “The Draco Approach to Constructing Software from Reusable Components.” In *Proceedings of the Workshop on Reusability in Programming.* ITT Programming, Stratford CT, September 1983.

[Perr92a] Perry, D., A. Wolf. “Foundations for the Study of Software Architecture.” *ACM Software Engineering Notes*, Vol. 17, #4, October 1992.

[Pine93] Pine, J. *Mass Customization: The New Frontier in Business Competition.* Harvard Business School Press, Boston MA, 1993.

[Prah89] Prahalad, C., and G. Hamel. “Strategic Intent.” *Harvard Business Review*, May-June 1989.

[Prah90] Prahalad, C., and G. Hamel. “The Core Competence of the Corporation.” *Harvard Business Review*, May-June 1990.

[Prie91a] Prieto-Diaz, R. Reuse Library Process Model. IBM STARS Technical Report CDRL 03041-002, STARS Technology Center, Arlington VA, July 1991.

[Prie91b] Prieto-Diaz, R., G. Arango. “Domain Analysis Concepts and Research Directions.” In *Domain Analysis and Software Systems Modeling*, R. Prieto-Diaz and G. Arango, ed., IEEE Computer Society Press, 1991.

[ROSE93a]STARS. The Reuse-Oriented Software Evolution (ROSE) Process Model, Version 0.5. Unisys STARS Technical Report STARS-UC-05155/001/00, STARS Technology Center, Arlington VA, July 1993.

[Sche87a] Schein, E. *The Clinical Perspective in Fieldwork.* Qualitative Research Methods Series 5, Sage, Newbury Park CA, 1987.

[Sche87b] Schein, E. *Process Consultation, Vol. 2: Lessons for Managers and Consultants.* Addison-Wesley, Reading MA, 1987.

[Schw91] Schwartz, P. *The Art of the Long View, Planning for the Future in an Uncertain World.* Doubleday/Currency, New York NY, 1991.

[Schw93] Schwartzman, H. *Ethnography in Organizations.* Qualitative Research Methods Series 27, Sage, Newbury Park CA, 1993.

[Seng90] Senge, P. *The Fifth Discipline.* Doubleday/Currency, New York NY, 1990.

[Seng94] Senge, P., et al. *The Fifth Discipline Fieldbook: Strategies and Tools for Building a Learning Organization.* Doubleday/Currency, New York NY, 1994.

[Shaw94a] Shaw, M. “Making Choices: A Comparison of Styles for Software Architecture.” Unpublished manuscript, Carnegie-Mellon University, Pittsburgh PA, May 1994.

[Shaw95a] Shaw, M, et al. “Abstractions for Software Architectures and Tools to Support Them.” To appear in *IEEE Transactions on Software Engineering*, 1995.

[Shri90] Shrivastva, S. and D. Cooperrider. *Appreciative Management and Leadership.* Jossey-Bass, San Francisco CA, 1990.

[Simo91a] * Simos, M. “The Growing of an Organon: A Hybrid Knowledge-Based Technology and Methodology for Software Reuse.” In *Domain Analysis and Software Systems Modeling*, R. Prieto-Diaz and G. Arango, ed., IEEE Computer Society Press, 1991.

[Simo91b] * Simos, M., “Navigating Through Soundspace: Modeling the Sound Domain At Real World.” In *Proceedings of the 4th Annual Workshop on Software Reuse*, Herndon VA, November 1991.

[Simo94a] * Simos, M., “Juggling in Free Fall: Uncertainty Management Aspects of Domain Analysis Methods.” In *Proceedings of the 5th International Conference on Information Processing and Management of Uncertainty in Knowledge-Based Systems*, Springer-Verlag, July 1994.

[Sold89a] Solderitsch, J., K. Wallnau, J. Thalhamer. “Constructing Domain-Specific Ada Reuse Libraries.” In *Proceedings of the 7th Annual National Conference on Ada Technology*, March 1989.

[Spra79a] Spradley, J. *The Ethnographic Interview.* Holt, Rinehart, and Winston, New York NY, 1979.

[Spra79b] Spradley, J. *Ethnographic Observation.* Holt, Rinehart, and Winston, New York NY, 1979.

[Tris83] Trist, E. "Referent Organizations and the Development of Interorganizational Domains." *Human Relations*, Vol. 36, No. 13, pp. 269-84, 1983.

[Unis88a] * Unisys. Reusability Library Framework AdaKNET and AdaTAU Design Report. PAO D4705-CV-880601-1, Unisys Defense Systems, System Development Group, Paoli PA, 1988.

[VCOE92a]Virginia Center of Excellence for Software Reuse andTechnology Transfer (VCOE). Domain Engineering Guidebook. Software Productivity Consortium, Herndon VA, SPC-92019-CMC, Version 01.00.03, December 1992.

[VCOE92b]Virginia Center of Excellence for Software Reuse andTechnology Transfer (VCOE). Reuse Adoption Guidebook. Software Productivity Consortium, Herndon VA, SPC-92051-CMC, November 1992.

[Wart92a] Wartik, S., R. Prieto-Diaz. “Criteria for Comparing Domain Analysis Approaches.” *International Journal of Software Engineering and Knowledge Engineering*, September 1992.

[Weis92] Weisbord, M. *Discovering Common Ground.* Berrett-Koehler, San Francisco CA, 1992.

[Wick94a] * Wickman, G., J. Solderitsch, M. Simos. “A Systematic Software Reuse Program Based on an Architecture-Centric Domain Analysis.” In *Proceedings of the 6th Annual Software Technology Conference*, Salt Lake City UT, April 1994.

[Wino87] Winograd, T., and F. Flores. *Understanding Computers and Cognition: A New Foundation for Design.* Addison-Wesley, Reading MA, 1987.

[Zubo88] Zuboff, S. *In the Age of the Smart Machine.* Basic Books, New York NY, 1988.